Analysis of Recombination in Molecular Sequence Data

Dissertation

zur Erlangung des Grades
des Doktors der Naturwissenschaften (Dr. rer. nat.)
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes
von

Jochen Maydt

Saarbrücken
2008

Tag des Kolloquiums:	16. Dezember 2008
Dekan:	Prof. Dr. Joachim Weickert
Berichterstatter:	Prof. Dr. Dr. Thomas Lengauer Prof. Dr. Jotun Hein

Abstract

We present the new and fast method Recco for analyzing a multiple alignment regarding recombination. Recco is based on a dynamic program that explains one sequence in the alignment with the other sequences using mutation and recombination. The dynamic program allows for an intuitive visualization of the optimal solution and also introduces a parameter α controlling the number of recombinations in the solution. Recco performs a parametric analysis regarding α and orders all pareto-optimal solutions by increasing number of recombinations. α is also directly related to the Savings value, a quantitative and intuitive measure for the preference of recombination in the solution. The Savings value and the solutions have a simple interpretation regarding the ancestry of the sequences in the alignment and it is usually easy to understand the output of the method. The distribution of the Savings value for non-recombining alignments is estimated by processing column permutations of the alignment and *p*-values are provided for recombination in the alignment, in a sequence and at a breakpoint position. Recco also uses the *p*-values to suggest a single solution, or recombinant structure, for the explained sequence. Recco is validated on a large set of simulated alignments and has a recombination detection performance superior to all current methods. The analysis of real alignments confirmed that Recco is among the best methods for recombination analysis and further supported that Recco is very intuitive compared to other methods.

Kurzfassung

Wir präsentieren Recco, eine neue und schnelle Methode zur Analyse von Rekombinationen in multiplen Alignments. Recco basiert auf einem dynamischen Programm, welches eine Sequenz im Alignment durch die anderen Sequenzen im Alignment rekonstruiert, wobei die Operatoren Mutation und Rekombination erlaubt sind. Das dynamische Programm ermöglicht eine intuitive Visualisierung der optimalen Lösung und besitzt einen Parameter α, welcher die Anzahl der Rekombinationsereignisse in der optimalen Lösung steuert. Recco führt eine parametrische Analyse bezüglich des Parameters α durch, so dass alle pareto-optimalen Lösungen nach der Anzahl ihrer Rekombinationsereignisse sortiert werden können. α steht auch direkt in Beziehung mit dem sogenannten Savings-Wert, der die Neigung zum Einfügen von Rekombinationsereignissen in die optimale Lösung quantitativ und intuitiv bemisst. Der Savings-Wert und die optimalen Lösungen haben eine einfache Interpretation bezüglich der Historie der Sequenzen im Alignment, so dass es in der Regel leicht fällt, die Ausgabe von Recco zu verstehen. Recco schätzt die Verteilung des Savings-Werts für Alignments ohne Rekombinationen durch einen Permutationstest, der auf Spaltenpermutationen basiert. Dieses Verfahren resultiert in *p*-Werten für Rekombination im Alignment, in einer Sequenz und an jeder Position im Alignment. Basierend auf diesen *p*-Werten schlägt Recco eine optimale Lösung vor, als Schätzer für die rekombinante Struktur der erklärten Sequenz. Recco wurde auf einem großen Datensatz simulierter Alignments getestet und erzielte auf diesem Datensatz eine bessere Vorhersagegüte in Bezug auf das Erkennen von Alignments mit Rekombination als alle anderen aktuellen Verfahren. Die Analyse von realen Datensätzen bestätigte, dass Recco zu den besten Methoden für die Rekombinationsanalyse gehört und im Vergleich zu anderen Methoden oft leichter verständliche Resultate liefert.

Acknowledgements

I would like to thank Prof. Thomas Lengauer for giving me the opportunity to work in his group, for his supervision and his continuous support. His input and knowledge on optimization techniques was particularly valuable for this thesis. I am also indebted to Prof. Jotun Hein for his kind willingness to review my thesis.

I am grateful to Prof. Andreas Meyerhans for sharing his extensive knowledge on recombination in biology and on recombination analysis in general. Prof. Jens Mayer and Aline Flockerzi conducted the analysis of HERV sequences with Recco together with me. Their feedback and the insights gathered during this analysis were invaluable to me and gave me a better understanding of gaps in recombination analysis.

I wish to thank all collaboration partners that have published together with me. I have worked closely with Francisco Domingues, Jörg Rahnenführer and Christoph Welsch and learned a lot about statistics and biology from them. I am happy that I had the chance to work together with them. André Altmann and Kasia Bozek did a great job proof-reading part of my thesis and gave me essential input for improving the exposition.

I am also grateful for the talks and discussions in our group that have deepened my understanding of evolution. The reading group with Niko Beerenwinkel and Tobias Sing was particularly stimulating and enlightening.

Finally and very importantly, I am deeply obliged to my family and to Heidi Rogosch. Without their continuous support and without them believing in me I would not have been able to finish this thesis.

Contents

1 Introduction

As sequencing technology advanced and generated more genetic sequence data, it became feasible to study the evolution of the genotype in addition to the evolution of the phenotype. The genetic sequence accumulates differences by operations such as mutation, recombination, insertion and deletion. Comparing genetic sequences therefore usually requires that we interpret differences as the result of these operations. Most of these operations take a single parental sequence as input and produce one or more child sequences. Recombination is different, though, as it combines the information of two or more parental sequences to produce the recombinant child sequence. For example, the two sequences AA and CC may produce the recombinant sequence AC (a more accurate definition is given in Chapter 2). In the following, we show why the analysis of sequences regarding recombination is important and which questions occur during the analysis of recombinant sequences.

Many methods for studying the evolution of a set of sequences assume that recombination does not occur and often produce wrong results otherwise. For example, phylogenetic tree inference assumes that the ancestry of the sequences can be described by a tree. But recombination violates this assumption, as recombination produces one (or more) child sequence from two or more parental sequences. The estimated tree then is not only wrong, but can also lead to wrong conclusions, such as exponential population growth or substitution rate heterogeneity among sites (Schierup and Hein 2000). It is therefore important to rule out the option of recombination in the alignment before these methods are applied. Hence, detecting recombination in an alignment is already an important task.

But it is also interesting to study the recombination process and the products of recombination itself. It is very likely that the recombination process plays an important role in evolution as it is extremely common in nature. Recombination, or at least the exchange and mixing of genetic information, may occur in principle during the reproduction of numerous organisms. Several proteins that allow for strand exchange and recombination are well-conserved across the three domains of life: eukarya, archea and eubacteria (Heyer, Ehmsen et al. 2003). Recombination in *homo sapiens* and many other sexually reproducing eukaryotes even requires complex biochemical machinery and is a strongly controlled process during cell division. Recombination can also occur during the replication cycle of several viral families, such as retroviruses (Mansky 1998; Negroni and Buc 2001) and flaviviruses (Gould and Solomon 2008). But do these organisms also generate recombinant sequences in nature?

Our current knowledge varies widely across different organisms. It was only recently discovered that C*andida albicans* may reproduce parasexually under rare circumstance and could therefore allow for recombination (Heitman 2006). The role of recombination is also unclear for several other pathogenic fungi. An interesting hypothesis is that these fungi have a clonal population under normal circumstances, but allow for sexual repro-

duction if environmental conditions change (Heitman 2006). Recombination plays an important role in shaping the genetic diversity in the population of many other pathogens. *Plasmodium falciparum*, one of the protozoans causing malaria, is well known to recombine frequently and shows large differences in recombination rates between subpopulations (Mu, Awadalla et al. 2005). Recombination frequently occurs in several bacteria and is probably the most important force shaping the genetic diversity in *Escherichia coli*, *Neisseria meningitidis* and other bacteria (Spratt, Hanage et al. 2001). Recombination also increases the speed of *Escherichia coli* adapting to adverse conditions *in vitro* (Cooper 2007). HIV, the causative agent of the AIDS pandemic is particularly well studied. The process of recombination is at least partially understood at the molecular level (An and Telesnitsky 2002) and several hypotheses about preferred recombination locations exist (Negroni and Buc 2001; Baird, Gao et al. 2006). The rate of recombination in HIV is about ten to one hundred times larger than the rate of mutations *in vitro* (Jetzt, Yu et al. 2000; Levy, Aldrovandi et al. 2004) and the prerequisites for frequent and visible recombination are also met in nature (Jung, Maier et al. 2002). Recombinant forms are common in HIV and play an important role in the pandemic (Kijak and McCutchan 2005). The role of recombination in the evolution of HIV is controversial, though. Theoretical studies show that, in general, recombination is only beneficial under certain selective conditions and for a certain range of effective population size (Althaus and Bonhoeffer 2005; Fraser 2005; Suryavanshi and Dixit 2007). But it is unclear whether these selective conditions are met for HIV *in vivo*. Some researchers argue against this and propose that recombination merely allows HIV to recover from defects in the genome (Bonhoeffer, Chappey et al. 2004). However, *in vitro* experiments showed that recombination facilitates the emergence of drug resistance (Moutouh, Corbeil et al. 1996). Finally, recombination is an important aspect in the evolution of the human genome and has been a matter of active research for decades. Genetic maps measure the distance along a chromosome as the probability of recombination between two loci and are important for assembling genomes and for studying disease-marker associations (Kong, Gudbjartsson et al. 2002). Recently, recombination hotspots have been discovered (Jeffreys, Kauppi et al. 2001) and are estimated to account for about 60% of recombination in the human genome (Consortium, Frazer et al. 2007). These findings along with high-throughput genotyping technologies allow for studying the connection between genotype and phenotype in humans to an unprecedented scale and greatly advance our understanding of genetic diseases in humans.

The high prevalence of recombination in nature calls for specialized methods that allow for analyzing recombination in sequence data. Depending on our knowledge of the recombination process in an organism and on the goal of the analysis, there are many different questions we would like to address. If an organism allows for recombination, what is the rate of recombination and the distribution of recombination breakpoints across the genome? What is the impact of recombination on the evolution of the organism? For ex-

ample, do recombinant sequences sometimes have a higher fitness and does recombination therefore speed up adaptation to changing environmental conditions? Which genotypes or phenotypes were combined for successful recombinant sequences?

For some organisms, such as *Candida albicans*, it is unclear whether recombination occurs in nature. A method for detecting potential recombination events in the studied sequence set can either find evidence for recombination or confirm that standard phylogenetic methods apply. For other organisms and viruses, such as HIV, it is clear that recombination occurs frequently in nature, but the role of recombination for adapting to new selective pressures is poorly understood. In this case, it is often informative to determine the recombinant sequences and study these in more detail. For example, the geographic distribution of the different subtypes and recombinant forms of HIV allows for monitoring the progress of the pandemic. The composition of the recombinant forms regarding the parental sequences is also interesting, as it may help to explain the success of the recombinant forms in the pandemic. The composition of recombinant sequences is also interesting for intra-patient data, as multi-drug resistance might arise by recombination of several single drug resistant sequences. Other types of analyses are more relevant for the human genome. Methods from population genetics can estimate the recombination rate for each locus and can identify potential recombination hotspots (McVean, Myers et al. 2004). Some methods even try to infer the genealogy with recombination for a set of sequences and use this information to increase the accuracy of association studies (Minichiello and Durbin 2006; Wu 2007).

In this thesis, we describe Recco (Maydt and Lengauer 2006), a novel method for the analysis of recombination in a set of sequences. Recco mainly addresses four questions: 1) Is there recombination in the dataset? 2) Which sequence is the recombinant? 3) Where are the recombination breakpoints? 4) And what is the composition of the recombinant sequence? To be more specific, Recco infers potential recombination events in the alignment and returns a point estimate for a recent part of the genealogy. This type of analysis is particularly popular for studying the evolution of HIV and other viral pathogens. As already mentioned before, there are many other types of analyses addressing recombination and we do not cover them in this thesis. For example, population genetic methods estimate parameters such as the recombination rate or the distribution of recombination breakpoints across the genome (Kuhner, Yamato et al. 2000). Several other methods implicitly allow for recombination in the dataset, but do not have the objective of studying the recombination process itself. Methods for association analysis, for example, can predict the position of a disease marker even more accurately if there is recombination in the dataset.

The study of recombination may eventually assist in resolving the enduring paradox of sex and recombination (Otto and Lenormand 2002): why is sex and recombination so ubiquitous in nature, despite its apparent cost? Sexual reproduction requires the individual to invest much more effort into reproduction. Finding a mate obviously takes time and

energy. The unit of sexual reproduction is the couple and not the individual – if the reproductive rate in the population should stay the same, each couple has to produce twice as many offspring as a single individual. Finally, mixing the genome with another individual does not necessarily lead to a selective advantage. And still, nature has evolved and maintained sex and recombination for at least a billion years. Is sexual reproduction simply an "infective" reproduction mode that spreads through the population (Otto and Lenormand 2002) and then selects the conditions for its own preservation (Azevedo, Lohaus et al. 2006)? Or does sexual reproduction confer a selective advantage as natural populations are finite in size and as recombination can reduce the effects of linkage and drift in finite populations (Otto and Lenormand 2002)?

1.1 Outline

Chapter 2 describes the replication cycle in several species as it is the process that causes genetic changes during evolution. In particular, the current knowledge about the recombination process of HIV-1, HERVs and homo sapiens is summarized, paving the road for an appropriate model of recombination. We also discuss processes similar to recombination and precisely define the type of recombination that we model in the following. We draw several conclusions from biological reality and motivate, for example, that it is appropriate to use multiple sequence alignments as input for recombination analysis.

Chapter 3 approaches evolution from a theoretical point of view. We show that the neutral theory of evolution is a fundamental assumption for the analysis of recombination. Then, we describe the coalescent with recombination as it introduces the ancestral recombination graph (ARG), an accurate model for representing the ancestry of sequences subject to recombination. The ARG contains all information on past recombination events and is important for two reasons: it serves as a model for simulating sequences for validation, and it leads to fundamental insights regarding the analysis of recombination. For example, some types of recombination events cannot be reconstructed from sequence data.

Chapter 4 defines the objectives and the results we expect from recombination analysis in more detail and provides a summary of many methods for recombination analysis. We also discuss several strategies that are common for these methods. The methods and strategies are then compared regarding their advantages and limitations for the analysis of recombination.

Chapter 5 introduces Recco, the method for recombination analysis developed in this thesis. We describe the fundamental optimization problem of Recco and the structure of the optimal solutions. Two types of sensitivity analysis then allow for assessing the robustness of these solutions. Particularly one type, the parametric analysis, provides much more information on the recombination signal and allows for a fully automated recombination analysis of sequence alignments. Furthermore, we show how to address most objectives of recombination analysis and derive p-values to assess the robustness of our es-

timates. Finally, we compare Recco to other methods for recombination analysis and describes the unique aspects of our approach.

Like some other methods for recombination analysis, Recco may not perform well if there are large gap regions in the alignment. Chapter 6 first discusses several common strategies for treating gaps in phylogenetic analysis and then describes the problem of adequately treating gaps in recombination analysis and for Recco in particular. Treating gaps as missing data is particularly attractive, but is difficult to achieve for Recco. We propose a different strategy that requires some manual interaction for difficult recombination scenarios. A more detailed analysis of gap cost functions and their impact on the output of Recco also provides deeper insights about reasonable gap costs.

Chapter 7 compares Recco with several other methods for analyzing recombination on simulated and on real alignments. Recco usually has a higher power for detecting recombination, the recombinant sequence or the breakpoint than other methods on simulated alignments. We also test Recco on HIV-1 and HERV alignments. Recco tends to predict recombination events more conservatively than other methods for HIV-1 subtyping. The analysis of the HERV alignments is particularly challenging as they contain several large gaps.

Finally, Chapter 8 discusses several avenues for further research, both for recombination analysis in general and for Recco, in particular. Some extensions for Recco are discussed in more detail, such as a strategy to reconstruct the ARG heuristically.

2 Evolution From a Biological Perspective

Recombination occurs during the replication cycle of organisms and can generate new genetic sequences. But the evidence that recombination leaves in the genetic sequences depends on many aspects of evolution and not only on the recombination process itself. For example, selection may clear recombinant sequences from the population, such that we do not observe recombination in practice.

Hence, we describe the evolutionary process briefly from an abstract point of view and clarify the interactions between the replication cycle, fitness and selection. The replication cycle and in particular the recombination process in two viral species, human immunodeficiency virus (HIV) and human endogenous retroviruses (HERVs), are described in more detail. We also use sequence data from both species for validation in chapter 7. The mutation process is not described in detail as the mathematical models for the mutation process are close enough to reality for our purpose (see section 3.2). Furthermore, selection and the relationship between phenotype and genotype are only briefly addressed, not only because they are far too complex, but also because methods for analyzing recombination commonly assume that most mutations are neutral (see section 3.1.1).

The recombination process is particularly well studied for HIV. The prerequisites that are necessary to produce recombination visible in sampled sequences are also well known for HIV and are good examples for the disparity between the potential of recombination and visible recombination. The recombination process in HIV also serves as a prototype for modeling recombination in Recco. The model of the recombination process is very simple, but applies to recombination in many other species as well. We also present several processes that are similar to recombination and describe their unique characteristics. For example, gene conversion is particularly relevant for eukaryotic evolution and can also be studied by methods for recombination analysis. Based on the analysis of the recombination process in biology, we then draw several conclusions that are directly relevant for the analysis of recombination. For example, we can usually use a multiple alignment as an input for the analysis without imposing unrealistic assumptions.

2.1 Introduction to Evolution

Evolution in a broad sense is the principle of continuing adaptation, based on the introduction of small modifications and selection for the best alternative. Evolution proceeds by repeatedly applying the operations of selection and replication to the entities in a population. Selection ensures that entities with a higher fitness have a higher probability of replicating and become more abundant in the population. Replication, or the copy process, introduces small modifications, such that the fitness of the copies can differ from that of the template. The copies are then subject to selection and the process is repeated. A consequence of the process of evolution is that the fitness of the population usually increases and the population is better adapted to the environment. Again, selection usually

does not force modifications to occur if they are beneficial, but only increases the survival probability once they occur in the population. The evolution of living organisms is more complex, as the copy process is also subject to selection. For example, the copy process could be selected such that it modifies a certain gene if some environmental condition is present. But evolution of the copy process is usually rather simple: for example, the mutation rate per nucleotide base differs by several orders of magnitude in different organisms and is probably the result of selection (Drake, Charlesworth et al. 1998; Sniegowski, Gerrish et al. 2000).

The effect of selection is commonly classified into positive and negative selection. Positive (negative) selection occurs if a modification is associated to a fitness that is higher (lower) than the average fitness in the population. As a consequence, positive (negative) selection leads to an increased (decreased) frequency of the modification in the population. Negative selection can also cause different mutation rates among genes in the same organism. If a gene is important, many modifications are negative or even lethal for the individual and therefore negative selection prevents these modifications from persisting in the population. The result is that the observed mutation rate of important genes can be lower than the mutation rate of other genes, even though the underlying mutation rate is the same.

Evolution can be described in rather abstract terms as above and is generic enough to encompass not only the evolution of biological individuals, but also the evolution of ideas or information. However, the process of evolution of an organism or a virus is inherently bound to the phenotype that interacts with the environment and to the cellular machines that store, transport and duplicate the genomic information. It is important to understand these aspects in some detail, as a thorough analysis of genetic sequences is impossible otherwise. Hence, this chapter only covers the evolution of genetic information such as RNA, DNA and protein sequences.

2.1 HIV Evolution and Biology

2.1.1 HIV Origin and Subtypes

The human immunodeficiency virus (HIV) is the cause of the global AIDS pandemic. It has evolved by cross-species transmission and adaptation of the simian immunodeficiency virus (SIV) in two ape species. HIV-1 originated from SIV in chimpanzees, while HIV-2 evolved from SIV in sooty mangabeys (Heeney, Dalgleish et al. 2006). At least three cross-species transmission events gave rise to genetically distinct groups in HIV-1: group M, group N and group O (Keele, Heuverswyn et al. 2006). Similarly, HIV-2 falls into group A to group H. The focus of this section is group M of HIV-1, as it is the cause for the AIDS pandemic and is of premier interest. Nevertheless, the replication cycle and many other molecular mechanisms are very similar in HIV-2 and in other retroviruses.

Phylogenetic analyses of group M revealed clusters of sequences that were termed subtypes A to K (Robertson, Anderson et al. 2000). These subtypes often occur

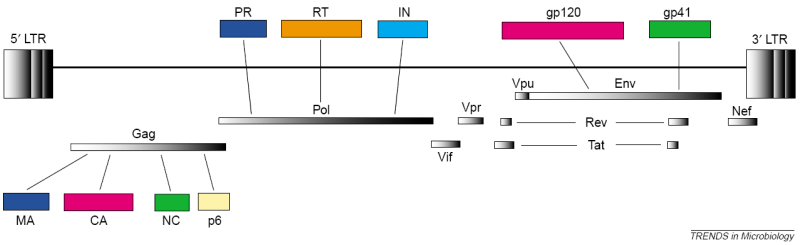


Figure 2.1: HIV-1 genome organization. The three major genes gag, pol and env are surrounded by the long terminal repeat regions. Each major gene encodes a protein precursor that is cleaved into several different proteins. See text for details. Reprinted from (Freed 2004) with kind permission from Elsevier.

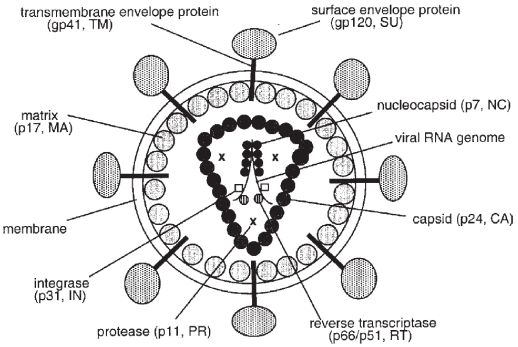


Figure 2.2: Schematic representation of the virion structure of HIV-1. The two viral genomes are packaged together with protease, reverse transcriptase and integrase into the capsid. Reprinted from (Freed 2001) with kind permission from Springer Science+Business Media.

predominantly in certain geographic regions: subtype B predominates in Europe, Australia, North and South America, subtype C in southern Africa and India and subtype A in East and Central Africa and East Europe (Thomson and Nájera 2005). Even though this suggests that the subtype structure is a result of founder effects, there is also evidence that some subtypes are due to incomplete sampling: several strains in Central Africa that were recently sequenced fall between previously described subtypes (Rambaut, Posada et al. 2004). Hence, the subtype structure in HIV-1 may get blurred as more sequences become available. The analysis of subtypes is further complicated because recombination between subtypes is possible, giving rise to circulating recombinant forms (CRFs) or unique recombinant forms. HIV-1 recombinant forms are estimated to account for 20% of worldwide HIV-1 infections today (Galetto and Negroni 2005). More recombinant forms were

found in regions where several diverse HIV-1 strains circulate, supporting the concept of geographic recombination hotspots (Thomson and Nájera 2005). This highlights the impact of recombination on HIV-1 genetic diversity and either suggests that recombination between subtypes is very frequent or that the CRFs are positively selected and exhibit a selective advantage over pure subtype strains.

Estimating the mosaic subtype composition of the recombinant forms of HIV-1 is often interesting and helps to understand the spread of the pandemic in more detail. Several methods have been developed for this purpose (see section 4.3.3), as traditional phylogenetic analysis cannot be used in the presence of recombination. The subtype composition of the recombinant forms is often further confirmed and analyzed manually. Consequently, the recombinant forms provide a good opportunity to validate methods for the analysis of recombination, such as Recco, the method developed in this thesis (see section 7.2.2).

2.1.2 HIV-1 Genes and Structure

A unique feature among retroviruses such as HIV is that they package two genomic RNAs into each virion. Each RNA genome of HIV-1 has a length of about 9.2kb and encodes the three major genes that are common to all retroviruses: gag (group-specific antigen), pol (polymerase) and env (envelope) (Freed 2001). The genes encode protein precursors that are cleaved by the protease of HIV-1 or by the cell machinery into several other functional proteins. The gag gene codes for matrix, capsid, nucleocapsid and other smaller proteins. These proteins drive the assembly and release of virus particles and encapsidate the viral RNA genome. The pol gene codes for protease, reverse transcriptase, and integrase. These proteins cleave viral proteins, perform reverse transcription of the viral RNA into DNA and integrate the viral DNA into the host cell genome. While gag is also synthesized individually, pol is only synthesized in conjunction with gag as the gag-pol polyprotein. Finally, the protein product of env is cleaved into the surface glycoprotein gp120 and the transmembrane glycoprotein gp41, which are necessary for the viral entry into the host cell. In addition, HIV-1 also codes for several other regulatory and accessory genes (Figure 2.1), such as tat and rev. The structure of an HIV-1 virion is displayed in Figure 2.2.

2.1.3 The Replication Cycle in HIV-1 (and Other Retroviruses)

The replication cycle of retroviruses is divided into an early phase and a late phase. The early phase encompasses all steps from viral entry to integration of the viral cDNA into the cell genome. The late phase starts with expression of the viral genes and ends with the release of new virions from the cell. Even though the replication cycle is very similar for many retroviruses, the following exposition is restricted to the replication cycle of HIV-1. The retroviral replication cycle is well understood and a lot more information can be found in several reviews (Freed 2001; Freed 2004; Nisole and Saïb 2004), and in (Rubbert, Behrens et al. 2007). In essence, the replication cycle of HIV-1 proceeds by the following steps (also see Figure 2.3):

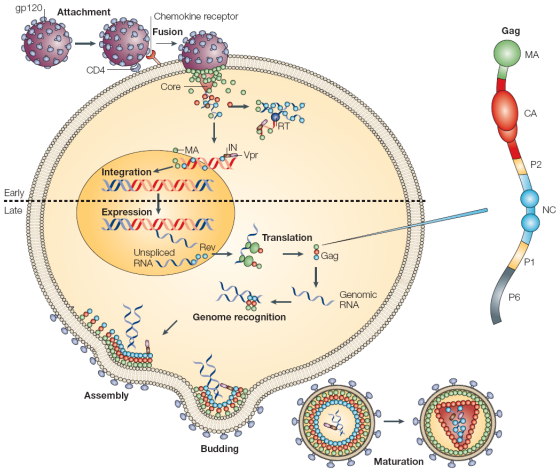


Figure 2.3: The steps of the retroviral replication cycle are described in the text. Reprinted by kind permission from Macmillan Publishers Ltd: Nature Reviews Microbiology (D'Souza and Summers 2005), copyright 2005.

Binding and Entry

After attachment to the target cell (mainly T-lymphocytes and macrophages), the surface protein gp120 recognizes CD4 on the target cells. Conformational changes then facilitate interaction of one of the coreceptors CXCR4 or CCR5 with the viral envelope. Finally, viral and cellular membranes are fused and the viral core is released into the host cell.

Uncoating and Reverse Transcription

After its release into the cytoplasm, the viral core starts disassembling, forming a reverse-transcription complex and later on a pre-integration complex. The reverse transcriptase (RT) copies the two viral RNA genomes, or templates, into a double stranded DNA. The DNA is usually not an exact copy: besides introducing point mutations, the RT can also switch from one template to the other and effectively produce a mosaic of both viral RNAs. The reverse transcription step is described in more detail in the next section, as it is the main source for genetic variation in HIV-1.

Nuclear Entry and Integration

The pre-integration complexes move to the cell nucleus by hijacking the cell machinery for cellular transport. Unlike other retroviruses, lentiviruses such as HIV-1 can actively cross the nuclear membrane and infect non-dividing cells. Integration of the viral DNA is

mediated by the integrase protein and occurs preferentially in genes that are highly transcribed by RNA Polymerase II.

Transcription and Export of mRNA

The transcriptional transactivator protein tat increases RNA synthesis of the HIV-1 provirus by more than two orders of magnitude and ensures efficient viral replication. The viral protein rev ensures the export of complete RNA transcripts from the nucleus and avoids that all transcripts are spliced by the cell before nuclear export. Translation then synthesizes the viral proteins.

Particle Formation

The precursor polyproteins gag and gagpol associate with the plasma membrane and build the viral particles through several interactions with each other, the plasma membrane and the viral RNA. The viral particles also incorporate other viral proteins, such as the envelope proteins and the protease.

Budding and Maturation

The process of budding releases the viral particles from the plasma membrane. During maturation the viral protease then cleaves gag and gagpol which leads to extensive protein rearrangements and formation of infectious particles.

The interaction of the virus with the cell machinery is even more involved than the description above would suggest. For example, the accessory gene vif (viral infectivity factor) inactivates the host cell protein APOBEC3G, a cytidine deaminase that interferes with reverse transcription (Freed 2004). Obviously, selection plays a vital role in the replication cycle of HIV-1 and adjusts all genes to reach high efficiency in a wide range of settings. But the success of HIV-1 also depends crucially on its capability of adapting to a fast changing environment. HIV-1 almost always causes persistent infections in spite of changing drug and immune pressure and even adapts to patient specific factors, such as HLA type (Ahlenstiel, Roomp et al. 2007; Bhattacharya, Daniels et al. 2007).

While selection targets the phenotype of the virus and determines its success in different environments, the phenotype of the virus depends on the genotype. Hence, selection can only work if the replication process introduces variation into the genome. In principle, the genome of HIV-1 can change during reverse transcription to the proviral DNA and during transcription of the provirus into genomic RNAs. The latter is part of the cellular machinery and has a very high accuracy. In comparison, the reverse transcription step introduces much more variation by mutation and recombination and is the driving force behind the evolution of HIV-1. The next section therefore only describes the reverse transcription step in more detail.

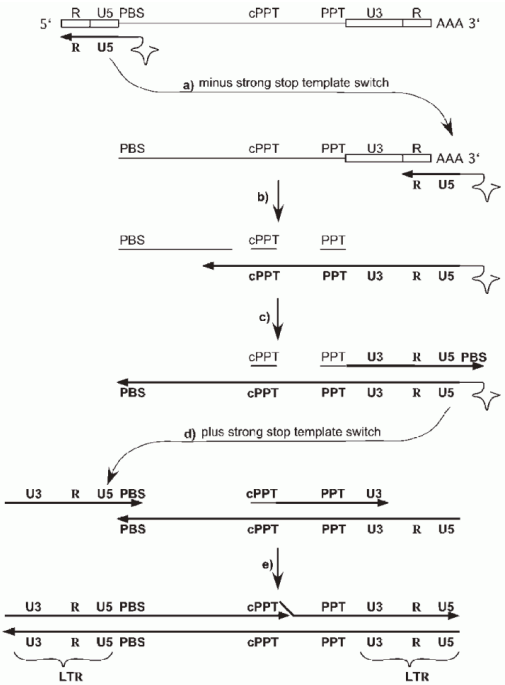


Figure 2.4: All steps of the reverse transcription process are shown, starting with the viral RNA and ending with the proviral double stranded DNA (shown as a bold line). From (An and Telesnitsky 2002), reprinted with permission from Publications Permanyer SL. **a)** Synthesis of the (-) strand DNA already involves a strong-stop template switch at the primer binding site (PBS). **b)** The RT digests the RNA template during DNA synthesis. **c),d)** Synthesis of the (+) strand DNA also involves a forced template switch at the central polypurine tract (cPPT).

2.1.3.1 Mechanisms of Recombination in HIV-1

There are several reviews about the recombination mechanism in HIV-1 and other retroviruses. The review by An and Telesnitsky (An and Telesnitsky 2002) also refers to experimental techniques for studying the recombination process. Negroni and Buc (Negroni and Buc 2001; Negroni and Buc 2001) and Galetto and Negroni (Galetto and Negroni 2005) provide more details on the putative process of template switching during reverse transcription in retroviruses.

Recombination occurs during the reverse transcription step when the reverse transcriptase (RT) enzyme switches from one template that it copies to another. Reverse transcription is a rather complex process and involves two basic steps in HIV-1 (see Figure 2.4). First,

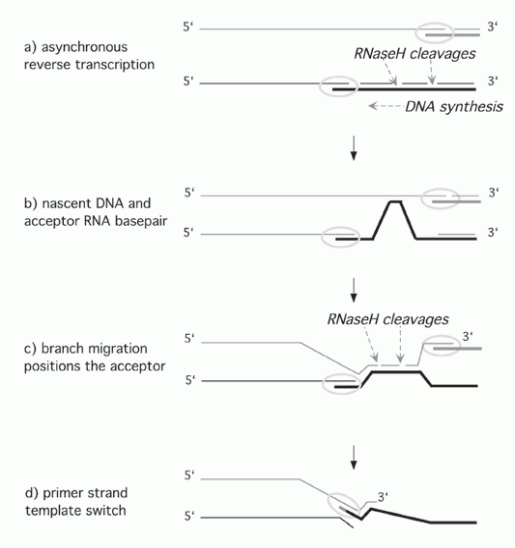


Figure 2.5: Template switching during (-) strand DNA synthesis follows a defined process. RNA is shown as a thin grey line and DNA is shown as a dark grey or black line. From (An and Telesnitsky 2002), reprinted with permission from Publications Permanyer SL. **a)** Reverse transcription can start simultaneously for both RNA strands and synthesize two nascent (-) strand DNAs. RNaseH cleaves and digests the copied RNA during DNA strand synthesis. **b)** The synthesis of the lower DNA strand is faster than the synthesis of upper DNA strand. This allows for the lower DNA strand to interact with the complementary upper RNA strand. **c)** RNaseH can now cleave the upper RNA strand, which in turn interrupts the synthesis of the upper DNA strand. **d)** The RT that is bound to the lower DNA strand may then switch templates and continue to copy the upper RNA of the virus, effectively producing a recombinant DNA.

the RT copies the genomic RNA into the (-) strand DNA and simultaneously digests the copied part of the RNA. As a result, the (-) strand DNA is a complement of the RNA at every position and matches A with T (or U in the RNA) and G with C. The (+) strand DNA is then synthesized from the (-) strand DNA and therefore encodes the original RNA sequence. The two DNA strands hybridize and form the double stranded proviral DNA. Both steps of reverse transcription involve several discontinuities where the RT dissociates from the template it copies and resumes copying at a different position, piecing together a complete copy of the viral genome (for details refer to (An and Telesnitsky 2002)). Recombination happens if the RT dissociates from the template and resumes copying on a different template. Even though recombination can take place during both steps of the proviral DNA synthesis, experimental data suggests that recombination during (-) strand DNA synthesis is much more frequent (An and Telesnitsky 2002). In the

following, only mechanisms that lead to recombination during (-) strand DNA synthesis are reviewed. Hence, we refer to the (-) strand DNA as the nascent DNA, to the RNA that is copied before recombination takes place as the donor RNA and to the RNA that is copied after the recombination event as the acceptor RNA.

Several mechanisms have been proposed to cause recombination during the synthesis of the (-) strand DNA, yet they all share the same basic principle: the RT dissociates from the donor RNA and continues polymerization of the nascent DNA using the acceptor RNA as a template (see Figure 2.5). An important requirement for the template switch is that the copied RNA is hydrolyzed by the RNaseH domain of the RT enzyme a few nucleotides behind the DNA polymerization site. This leaves the nascent DNA free to interact with other complementary sequences, for example the other genomic RNA packaged into the virion. As the DNA and the acceptor RNA form a complex, the acceptor RNA may displace the donor RNA and the RT continues to copy the acceptor RNA. While RNA degradation and hybridization of the nascent DNA to another RNA molecule are prerequisites for recombination, the actual site where the template switch occurs depends on the mechanism of recombination. Several mechanisms have been proposed:

Strong-Stop Strand Transfer

As mentioned above, the nascent DNA is never polymerized in a single run. Reverse transcription starts at the primer binding site close to the 5' end of the RNA and proceeds further towards the 5' end, thus only copying a tiny stretch of RNA. As the RT reaches the end of the RNA it dissociates from the RNA it copies (see Figure 2.4). The 5' and 3' end of the RNA genome encode the same repeat sequence. The nascent single stranded DNA can therefore transfer and anneal to the 3' end of any of the two RNAs and reverse transcription can then resume. The choice of the RNA to use as a copy template could be different from the RNA used to copy the 5' end and lead to a template switch. Unlike all other mechanisms of recombination, the recombination breakpoint for strong-stop strand transfer always occurs at the same position in the genome.

Forced Copy-Choice Strand Transfer

If the RNA template is not continuous and contains breaks, reverse transcription stops at the breaks. The same process as described for strong-stop strand transfer then causes RT to switch to another template and resume polymerization. While forced copy-choice strand transfer is an important mechanism to rescue proviral DNA synthesis in the presence of genome breaks, it does not seem to be the major cause for recombination in retroviruses. Experiments showed that the number of genome breaks is usually much smaller than the number of recombination breakpoints (Negroni and Buc 2001; Galetto and Negroni 2005).

Pause-Driven Strand Transfer

It seems plausible that the probability of a strand transfer increases if the acceptor RNA has more time to displace the donor RNA from the interaction with RT. Some sites in the HIV-1 genome are known to stall reverse transcription. In contrast to forced copy-choice, the obstacle is not enduring and reverse transcription can resume on the donor RNA without a template switch. Experimental evidence showed that at least one strong pause site in HIV-1 contributes significantly to efficient strand transfer at the pause site (Negroni and Buc 2001; Galetto and Negroni 2005). While there are some pause sites that correlate with strand transfer, this is not true in the majority of cases. Experiments in murine leukemia virus and *in vitro* showed that slowing down reverse transcription in general also leads to a higher frequency of template switches. The emerging picture today is that pausing might facilitate strand transfer independent of the underlying mechanism, but is not genuinely correlated with hotspots of recombination (Galetto and Negroni 2005). It is also unclear whether the template switch itself causes a pause in reverse transcription and therefore whether pausing is the consequence and not the cause of a template switch.

Secondary-Structure Mediated Strand Transfer

A putative model of recombination proposes that the secondary structure of the RNA itself triggers the template switch. To be more specific, as the RT proceeds through a hairpin structure in the RNA, the nascent DNA, the donor and the acceptor RNA may engage in a rather complex interaction that brings the acceptor RNA close to the catalytic site of RT. Details on the interactions inside the hairpin structure can be found in (Negroni and Buc 2001; Galetto and Negroni 2005). Secondary-structure mediated strand transfer occurs preferentially after the bulge of the hairpin. The hairpin structure may also slow down reverse transcription and lead to pause-driven strand transfer. However, secondary-structure mediated and pause-driven strand transfers are different, as the latter predicts a strand transfer at the base of the hairpin.

Despite many experiments involving the reverse transcription step in HIV-1 it is still not clear if one mechanism for strand transfer dominates. Another source of uncertainty is that experiments are often conducted in gammaretroviruses or murine leukemia virus and not in HIV-1. Even the recombination rate differs remarkably between these retroviruses and the results on the mechanisms of template switching may not always generalize to HIV-1.

An interesting observation is that the sequences involved in the template switch usually share a high sequence similarity around the recombination breakpoint, as the cDNA and the acceptor sequence have to hybridize before the template switch. Indeed, experiments *in vitro* have shown that the efficiency of template switches decreases about 100- to 1000-times if the two RNAs differ in the region surrounding the recombination breakpoint (An and Telesnitsky 2002). In gammaretroviruses, a stretch of 12 identical

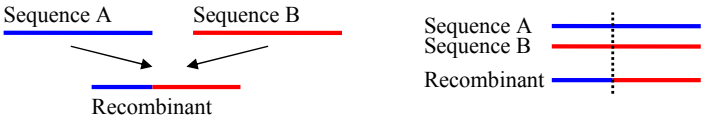


Figure 2.6: **(a)** Two sequences A and B, represented with different colors, recombine and produce a recombinant that is identical to A in the left part and identical to B for the rest of the sequence. **(b)** Both sequences usually have a high sequence identity around the recombination breakpoint. Consequently both sequences are usually homologous at the recombination breakpoint and are aligned correctly before recombination. Hence, they can be represented in a multiple alignment.

nucleotides between donor and acceptor RNA is sufficient to align the sequences accurately before the template switch. However, such a short stretch of sequence similarity lowers the frequency of the template switch. Hence, the donor and acceptor sequences usually share a high sequence similarity around the recombination breakpoint and it is very likely that the sequence donor and the acceptor are aligned globally before recombination. In this case, the product of recombination can be visualized in a multiple sequence alignment together with the donor and acceptor sequence (see Figure 2.6).

In summary, recombination in HIV-1 has a very simple structure: the recombinant is composed of the donor sequence in the first part up to the recombination breakpoint and the remainder of the recombinant is derived from the acceptor sequence. The recombinant is always composed as depicted in Figure 2.6 (a).

2.1.3.2 Visible Recombination

It is estimated that template switching occurs in HIV-1 at a rate of three to nine template switches per genome per replication round (Jetzt, Yu et al. 2000; Levy, Aldrovandi et al. 2004) and exceeds the estimated mutation rate of about 0.25 mutations per genome per replication round (Mansky and Temin 1995) by far. But template switching does not suffice to generate visible recombinants. If the donor and acceptor sequences are identical and correctly aligned before recombination, the result of recombination is the same as if no recombination occurred. Visible recombination can only occur if a host cell is infected by at least two genetically different HIV-1 strains and if two different RNAs are then co-packaged into the same viral particle. The heterozygous virion can then generate a recombinant upon infection of another cell.

Several aspects of the replication cycle of HIV-1 have the potential of interfering with and preventing multiple infections and co-packaging of different RNAs. First, the viral proteins vpu, nef and env interact with CD4 and the host cell after infection to stimulate degradation of CD4 and prevent its trafficking to the plasma membrane (Freed 2001). As HIV-1 depends on CD4 for cellular entry, it was believed that multiple infection of the

same host cell by HIV-1 would be rare. However, a recent study showed that 75%–80% of infected cells harbor more than one provirus and that the proviruses are genetically diverse (Jung, Maier et al. 2002). Second, co-packaging usually happens if two RNAs form a non-covalently linked dimer, initiated by a dimer initiation site near the 5' end of the HIV-1 genome. As dimerization may be ineffective or even impossible for different RNAs, co-packaging could preferentially pair highly similar sequences and inhibit visible recombination. But it was shown that two RNAs can still become co-packaged with a possibly lower efficiency, even if the dimer initiation site of the two RNAs are unable to interact properly (An and Telesnitsky 2002).

Hence, visible recombination occurs frequently in HIV-1, but the rate of visible recombination is quite different from the rate of template switching. Unfortunately, it is impossible to obtain an accurate estimate of the fraction of template switches that lead to visible recombination, as the data available are based on small sample sizes. A rough estimate is still possible. Let us assume that co-packaging occurs randomly and does not limit the generation of heterozygous viral particles. In this case, the number of heterozygotes is predicted by the Hardy-Weinberg equilibrium: if a cell is infected by two proviruses *A* and *B* and both are expressed with frequencies f_A and f_B, the probability of generating a heterozygous (or homozygous) virion is $2f_Af_B$ (or $f_A^2+f_B^2$). Assuming that both proviruses are expressed at similar levels and hence $f_A = f_B = 0.5$, half of the produced viral particles are heterozygous and could result in visible recombination. It is trivial to extend the argument to more than two proviruses. The probability of multiple infection by 1, 2, 3, >3 proviruses is about 20%, 15%, 28%, 37% for HIV-1 (Jung, Maier et al. 2002). Assuming that cells with more than three proviruses generate only heterozygotes, the average amount of generated heterozygotes is 15%/2+28%·3/8+37%=55%. Hence, even with these optimistic assumptions, only about half the viral particles are heterozygotes and lead to visible recombination.

2.2 Evolution of HERVs

HERVs, or human endogenous retroviruses, are remnants of exogenous retroviruses that have been integrated into the human germ line in the past. In contrast to exogenous retroviruses, HERVs are transmitted vertically from parents to their progeny and usually colonize every cell in the body. HERVs make up about 8% of the human genome and therefore play an important role in the evolution of the human genome (Bannert and Kurth 2004). Today, most HERVs are deeply fixed within the human and primate population. By continuous accumulation of mutations, the provirus of these older HERVs has become defective and is not transcribed nor does it code for proteins anymore. The genesis of these HERVs probably dates back to early mammalian evolution. Younger HERVs are exclusive to the primate lineage, and the youngest HERV family, HERV-K, started to endogenize about 45 million years ago. Few HERV-K loci still contain proviruses with intact ORFs, such that all three structural proteins gag, pol and env are transcribed. It is

therefore unclear to which degree these HERVs are still active and whether they are able to retrotranspose or form viral particles. There are two mechanisms driving the evolution of HERVs: the viral replication cycle (see section 2.1.3) and the human replication cycle (see section 2.3). As both mechanisms are described elsewhere, it is sufficient to explain the lifecycle of HERVs at a coarse scale.

2.2.1 Lifecycle of HERVs

The lifecycle of an HERV starts with infection of the host with an exogenous retrovirus and subsequent infection of germ line cells of the host (Löwer, Löwer et al. 1996; Gifford and Tristem 2003). If such an infected germ line cell gives rise to progeny, the provirus is part of every cell in the progeny and the retrovirus becomes an endogenous retrovirus, or HERV if the host is a human. As the provirus is still intact, it can create infectious virions that integrate not only into somatic cells, but also into germ line cells. Again, the human replication cycle continues and the progeny may contain a higher copy number of the HERV. This process of increasing copy number is called amplification and may continue for some time. Over time, the virus may loose its ability to create infectious virions and only amplifies by retrotransposition or similar processes. These processes usually reverse transcribe and integrate the genomic RNAs that are expressed in the cell, but do not need a viral particle for infecting other cells. However, it is clear that aggressive amplification is deleterious to the host, for example because the HERV may inactivate vital genes during its integration and cause cancer. In other words, harboring the HERV has a negative effect on the survival probability of the host. As a result, amplification should decrease over time and eventually stop. HERVs then only evolve as part of the human genome.

Evolution of an HERV involves selection pressure on both, the host and the HERV. The host may evolve mechanisms that prevent integration of retroviruses into the germ line, restrain HERV amplification or cause the deletion of the provirus from the genome. For example, the DNA can form a loop connecting the long-terminal repeats (LTRs) at both ends of the provirus such that a recombination removes the provirus from the host genome, leaving a solitary LTR as the remainder. Alternatively, mutations introduced into the provirus during the copy process of the human genome may also render the provirus defective and unable to amplify. On the other hand, the HERV provirus is exposed to two very different selection pressures: the provirus has to ensure its own survival just like every gene in the human genome, but also has to allow for the survival of the host. Hence, it may still be preferable for the HERV to occur in high copy number in the human genome and counteract the mechanisms that inactivate or clear the HERV from the genome. But in comparison to its exogenous phase, the HERV now does not have to build infectious particles anymore to be successful and spread in the human population. Its survival is directly linked to the fitness of the host – and it is sufficient that the host is successfully replicating. As a result, the HERV evolves a slower rate of amplification or even looses this ability altogether. Eventually, the HERV often ends up as junk DNA. A

few HERVs may acquire an important function for the host and ensure their survival. For example, one of the proviruses in the HERV-W family is important for forming the boundary layer between maternal and fetal tissue and plays an important role in the maternal-fetal exchange (Bannert and Kurth 2004).

In summary, HERV evolution is characterized by selection and rare random events. It is probably a rare event that a retrovirus successfully integrates into the germ cell which gives rise to progeny and that the integration event is non-lethal to the progeny. Duplication and modification of the HERV provirus occur during the retroviral replication cycle (see section 2.1.3) and during the human replication cycle (see section 2.3). The retroviral replication cycle occurs during the exogenous phase of the HERV and allows for a high mutation and recombination rate, but probably stops quite early in HERV evolution. In contrast, the human replication cycle has a much higher fidelity, but acts throughout time. It is unclear which process dominates the evolution of HERVs. The mutation rate of HERVs has been estimated to about 2.3×10^{-9} to 5×10^{-9} substitutions per site per year and is about the same as the rate of substitution in pseudogenes and non-coding regions of mammalian genes (Johnson and Coffin 1999).

2.3 Evolution of Homo Sapiens and Other Eukaryotes

The evolution of humans in the following is only relevant to the extent that the recombination process is described accurately. Humans and many other eukaryotes have a diploid set of chromosomes in each cell, where one haploid set of chromosomes came from the mother and the other set came from the father. During meiosis, the diploid chromosome set is doubled and separated into four gametes with a haploid chromosome set. As the chromosomes are separated, two homologous chromosomes may become aligned and produce two recombinant chromosomes upon separation. The process of recombination is assisted by several cellular proteins, but the product of recombination mostly depends on the Holliday junction. Finally, as the gametes of the mother and the father merge to become the zygote of a child, a diploid chromosome set is created again.

2.3.1 The Holliday Junction

In many organisms, such as humans and other eukaryotes, recombination involves an intermediate Holliday junction between two double-stranded DNA molecules (see Figure 2.7) (Lodish, Berk et al. 2001; Heyer, Ehmsen et al. 2003). The Holliday junction forms when two homologous double-stranded DNA molecules become aligned and break at least partially. The strands are exchanged between the DNA molecules and the cuts are joined to the other chromosome, giving rise to the Holliday junction. The branch point of the Holliday junction can then move easily and create a region of heteroduplex DNA. To be more specific, heteroduplex DNA occurs whenever a strand of one double-stranded DNA hybridize to a strand of another double-stranded DNA. The Holliday junction is then resolved by cutting one strand of each DNA and joining them appropriately.

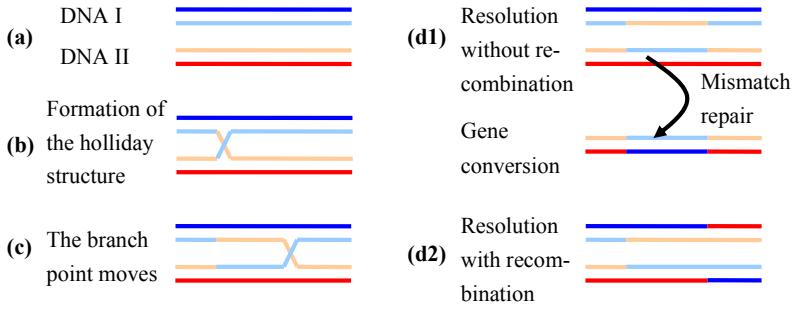


Figure 2.7: The Holliday junction and the recombination process, adopted from (Lodish, Berk et al. 2001). **(a), (b)** Strand breaks in the DNA molecules lead to the formation of a Holliday junction. **(c)** As the branch point moves, the heteroduplex region grows. **(d1)** The Holliday junction is resolved such that it causes no recombination. The heteroduplex regions may still lead to gene conversion when it is resolved as is shown exemplarily in the Figure. **(d2)** The Holliday junction is resolved such that it causes recombination.

Resolution occurs in two different ways, one leading to recombination together with some heteroduplex region at the recombination breakpoint and the other leading only to a heteroduplex region. Other, more sophisticated models for the formation and resolution of the Holliday junction exist, as well as models for recombination that do not involve the classical Holliday junction (Heyer, Ehmsen et al. 2003). However, in contrast to the models in HIV-1, all models predict that some heteroduplex DNA can be formed.

The mismatch repair system recognizes the heteroduplex DNA and selects one strand of the double-stranded DNA as a template for correcting the other. Consequently, both strands are complementary again. In some cases the selected template strand did not belong to the original double-stranded DNA molecule, and the result is a gene conversion (see Figure 2.7). In essence, gene conversion transfers one allele from one chromosome to the other and may change for example the alleles *Aa* at one locus to *AA* or *aa*. In contrast to recombination, gene conversion results in an asymmetric exchange of a small DNA region.

Besides the possibility of gene conversion, there are several other minor differences between recombination in eukaryotes and recombination in HIV-1. Recombination in eukaryotes is symmetric: both DNA molecules exchange parts of their information, such that both original DNA molecules could be reconstructed from the recombinants. But the symmetry hardly makes a difference, as only one of the two DNA molecules is passed on to the progeny. Recombination hotspots are likely to depend on other parameters and sequence motifs than in retroviruses, because the proteins involved in the process are very

different. Finally, the product of gene conversion resembles a recombination with two crossovers in close vicinity.

2.4 Other Mechanisms That Mix Genes and Genomes

There are several other mechanisms that combine the genetic information of two or more ancestors, but do not result in a recombination as described so far. It is important to introduce and understand these mechanisms, as they are often not well defined in the literature regarding methods for analyzing recombination. In addition, some methods refer to one or several of these mechanisms in conjunction with recombination.

Horizontal gene transfer, or lateral gene transfer, refers to a mechanism for acquiring novel DNA and is a major source of genome innovation in bacteria and other prokaryotes (Gogarten and Townsend 2005). There are bacteria that can directly take up raw DNA from their environment, integrate it into their genome and pass the DNA on to their progeny in a process called natural transformation (Thomas and Nielsen 2005). Similarly, conjugate transfer comprises the exchange of plasmids between bacteria and can lead, for example, to resistance to antibiotics. Finally, transduction refers to a gene transfer mediated by viruses, for example bacteriophages, that package some of the host's genes into their progeny by chance and reinfect other bacteria. Hence, horizontal gene transfer occurs by several different processes, but all lead to the exchange of genes and possibly integration into the host's genome.

Hybridization often refers to a crossing between two species such that the hybrid has one chromosome set of each parental species (Mallet 2007). The hybrid, for example a mule, is often infertile and unable to reproduce, because the chromosomes cannot pair properly during meiosis. However, if the chromosome set from each species is duplicated (allopolyploidy), pairing during meiosis is possible and the hybrid can be fertile. Other mechanisms can also lead to hybridization. Overall, hybridization is an important strategy for speciation and adaptive radiation, in particular in plants.

Reticulation or reticulate events comprise a diverse set of evolutionary processes, including hybridization, horizontal gene transfer, recombination, or gene duplication and loss (Huson and Bryant 2006). The unifying aspect of these events is that they lead to contradictory evidence concerning the ancestral relationship: while some parts of the genome are derived from one species or ancestor, other parts are not. The term reticulation is not clearly defined in literature and is also used to refer to network-like evolution. Some authors refer to reticulation as hybridization and gene transfer (Nakhleh, Warnow et al. 2005), others even refer to it as hybridization only (McBreen and Lockhart 2006).

Horizontal gene transfer and hybridization lead to the exchange of genetic information between organisms or species. Recombination, on the other hand, usually exchanges genetic information between homologous DNA molecules. Hence, in contrast to horizontal gene transfer and hybridization, recombination respects the linearity of the genome: the ordering of genes and alleles involved in recombination is important. Methods for analyzing recombination events can therefore use this aspect of recombination to achieve a higher accuracy or sensitivity.

2.5 Implications for the Analysis of Recombination

Analysis and inference need abstract models, but models without a foundation in the real world are usually irrelevant. The biology described above gives valuable input for the analysis of recombination in HIV-1 and other organisms, but the input is mostly qualitative. For example, the dominant mechanism of recombination in HIV-1 is unclear and even the rate of recombination cannot be accurately estimated. Both would be useful for developing a more accurate model for analyzing recombination, for example by providing an a priori probability of the recombination rate. But the risk of imposing a wrong prior is too high and may severely bias the results of the analysis: if the prior assumes a high recombination rate, the method actually infers a lot of spurious recombination events. On the other hand, if no informative prior is used, methods for analyzing recombination can provide estimates of the recombination rate or may even be used to find out which recombination mechanism dominates.

Nevertheless, biology contributes information that is important for the analysis of recombination. The knowledge that recombination does occur is already of high value for analyzing recombination by itself. Recombination must be inferred from mutation, as it only generates new combinations from existing mutation patterns. In other words, recombination events do not leave direct evidence in the sequence data. Hence, it is impossible in principle to infer that recombination does not occur.

Another interesting observation is that the replication cycle of HIV-1 causes an effect called negative interference: if two sequences are involved in one recombination event, the probability for additional recombination breakpoints is very high. The reason is simple: if two sequences recombine they must have been co-packaged into one virion which results in a high chance for additional template switches during reverse transcription.

Furthermore, intra-patient evolution and the evolution of the HIV-1 pandemic itself are qualitatively different. Even though co-infection of a patient with several different HIV-1 strains does occur, it is a rare event. The most prominent examples of recombination between patients are the CRFs of HIV-1. But even though the prevalence of CRFs in the HIV-1 pandemic is quite high, it does not indicate that CRFs are frequently generated. A better estimate is the number of distinct CRFs in the population, as selection probably leads to a high prevalence once a successful recombinant form arose in the population. The impact of recombination on intra-patient evolution is much more pronounced than on

```
A  1--456789
B  123456789
C  12345--89
```

Figure 2.8: Equal recombination is not a contradiction to insertions and deletions. The depicted alignment represents homologous loci or nucleotides by using the same number and indicates sequence similarity by color. Sequences A, B and C are different and represented by different colors. The recombinant sequence R is composed of sequences A and B and inherits the deletion of sequence A.

```
A  12345    A  12345        A  1234--5    A  12345
B  12345    C  1--45        B  1234--5    B  12345
                            C  1--4--5     C  1--45
C  1--45    D  14345        D  1--4345     D  14345
```

(a) **(b)** **(c)** **(d)**

Figure 2.9: Unequal recombination can lead to sequences that are incompatible with a multiple alignment. As in Figure 2.8, loci are numbered and colors represent sequence similarity. **(a)** The alignment contains a gap, because sequence C is the result of an unequal recombination between A and B. **(b)** An unequal recombination between C and A leads to sequence D. The second and the third nucleotide of sequence D contradict the ordering of the sequence, and a consistent multiple alignment is not possible. **(c)** A multiple alignment that aligns the first '4' in sequence D correctly. **(d)** The other option is to correctly align the second '4' in sequence D correctly, but misalign the first one.

the evolution between patients, as co-packaging of different strains occurs in every patient. Hence, methods that analyze sequences of different patients can ignore the effect of recombination to some extent.

2.5.1 Multiple Alignments and the Analysis of Recombination

Based on the exposition of the recombination process in HIV-1 and in humans we can distinguish three different kinds of recombination events, depending on the position of the breakpoint in the donor and acceptor sequence and on the local homology between sequences. If the recombination breakpoint occurs at the same genome position for both sequences, it interchanges homologous nucleotides and usually also involves sequences that are highly similar. We refer to this type of recombination as equal recombination in the following. Unequal recombination refers to a recombination event that occurs at different genome positions in the donor and acceptor sequence. A simple consequence is

that the product of recombination contains an insertion or a deletion. Unequal recombination can be classified further depending whether the donor and acceptor sequences are locally similar or locally dissimilar.

A particularly important observation is that most recombination events in HIV-1 and in eukaryotes are caused by equal recombination. As described before for humans and for HIV-1, unequal recombination is very inefficient if the involved sequences are dissimilar at the recombination breakpoint and therefore only happens very rarely, if at all. Unequal recombination involving locally similar sequences is much more efficient. The probability of such an event is still quite low, as there are not many different regions that share a longer stretch of identical nucleotides. Furthermore, if unequal recombination results in frame shifts or deletion of an important region, the genetic sequence is usually not viable and is quickly cleared from the population by negative selection. Hence, most observed recombination events must be due to equal recombination.

Equal recombination does not exclude insertion and deletion events. It is only required that the breakpoint position in the donor and the acceptor sequence occur at the same position in the genome, i.e. in the same column of a global pairwise alignment of both sequences. It is not possible to shuffle the homologous positions of a sequence if this restriction is satisfied. As a consequence, a multiple alignment exists and the recombination breakpoint always occurs between two columns in the multiple alignment, as shown in Figure 2.8. In contrast, unequal recombination allows to reorder a sequence, such that a consistent multiple alignment cannot be constructed (Wang, Ma et al. 2000) (see Figure 2.9). An analysis of such a sequence dataset is considerably more complicated and is not covered in this thesis.

It is usually possible to create a multiple alignment without the artifacts shown in Figure 2.9 in practice – at least if a comparatively small part of the genome is considered. Even if unequal recombination occurs in nature, it apparently does not affect the ordering of loci and is therefore identical to equal recombination with a mechanism that introduces insertion and deletion events.

3 Modeling Evolution

The biological process of evolution and the effect it has on the genetic sequence are difficult to understand without an abstract model of evolution. The process of evolution in viruses and in humans is very different and still, it is possible to describe the evolution of genetic sequences in both species using the same models. The most important abstraction introduced is that of neutral evolution (see section 3.1): genetic mutations do not change the fitness of an individual. In this case, the genetic sequence does not affect the success of the lineage, and the ancestral process is independent of the mutation process. To be more specific, a sequence dataset can be simulated by drawing a random ancestry first (see section 3.3) and then simulating the mutation pattern along the ancestry (see section 3.2). Hence, sequences share similarity by common descent and the ancestry can be estimated from sequence data.

A particularly useful model for the ancestral process is the coalescent, as it is a simple and robust approximation of many other ancestral processes. The coalescent with recombination introduces the ancestral recombination graph (ARG), an accurate representation of the ancestry of sequences subject to recombination. As the coalescent is a stochastic process, it also describes how to randomly draw ARGs. In combination with a random mutation process it is possible to generate artificial, but quite realistic sequence alignments for validation. But the ARG also helps to understand the complexity of inference if the ancestry includes recombination. If recombination occurs, each site may evolve according to a different evolutionary tree and the ancestry alternatively can be represented as a set of trees. However, even if these trees are known, they do not directly provide information on recombination events. We still have to infer recombination events from the trees indirectly as a subtree-prune-and-regraft operation. Finally, we distinguish different types of recombination events depending on their representation in the ARG. These aspects help to classify methods for the analysis of recombination in the next chapter and also allow understanding the limitations of each method in more detail.

3.1 The Neutral Theory and Selection in Sequence Analysis

The common understanding of evolution before the late 1960's suggested that the genetic composition of an organism was largely optimized and determined by the action of selection. Most genetic mutations would be deleterious for the individual and could not persist, while only few mutations would confer a selective advantage and become quickly fixed in the population. Here, mutation refers to a change on the level of an individual, whereas substitution occurs when a mutation spreads through the population. To be more specific, a substitution occurs when all other variants are eliminated and the mutation gets fixed in the population. Hence, it was assumed that selection is so ubiquitous and dominating that every amino acid substitution had some unique survival value for the organism. Random mutations could occur, but would be quickly cleared from the population by negative se-

lection and not become fixed. Polymorphisms in the population could only be maintained by balancing selection. Balancing selection occurs, for example, if heterozygotes are more fit than homozygotes and the average fitness in the population is larger with polymorphisms than without. In other words, the common perception was that selection for favorable traits (positive selection) adapts the phenotype to the environment and the selected phenotype determines the genetic sequence.

In contrast to this, even Darwin had already postulated that there could be variations that are neither positive nor negative and therefore would not be affected by selection. In 1968 Kimura challenged the view that the genome is mainly shaped by selection and proposed the neutral theory of molecular evolution (Kimura 1968; King and Jukes 1969). Based on the observation of amino acid substitution rates between mammalian species, he estimated that the nucleotide substitution rate amounts to at least one nucleotide substitution in the population every second year. Such a high rate of substitution could not be explained by balancing selection. Therefore, Kimura proposed that most mutations are almost neutral regarding selection and that positive selection is a rare event which only affects few sites. Neutral mutations fluctuate randomly in frequency and get fixed in the population by pure chance, an effect called random genetic drift. The neutral theory also allows for a substantial amount of deleterious mutations if their negative fitness effect is so strong that they are cleared rapidly from the population. It was assumed that the neutral mutations could occur at synonymous and non-coding DNA sites, but also at coding sites if the protein retains its functionality.

As sequence data became more and more abundant, the neutral theory was tested extensively on many datasets and deviations from the neutral theory were found. It is clear today that selection plays a role even for the evolution of synonymous sites and non-coding DNA if the population size is large (Chamary, Parmley et al. 2006). But even though most sites in the genome are subject to selective pressure, the selective pressure is often too weak to dominate genetic drift. The nearly neutral theory incorporated this observation and introduced the notion of nearly neutral mutations whose fate is determined by both, drift and selection (Ohta 2002). Interestingly, the strength of drift depends on the effective population size: it is weak in large populations and even small selective forces can dominate drift in this case.

The formulation of the neutral theory was a big step forward for the analysis of genetic sequences. The neutral theory assumes that the major force which shapes substitutions in the population is not selection and therefore the environment, but the random process of mutation and descent. The selectionist viewpoint and the neutral theory also differ fundamentally concerning the relevance of polymorphisms (Bamshad and Wooding 2003). The selectionist viewpoint assumes that polymorphisms are the signature of selection and are therefore functionally relevant. In contrast, the neutral theory assumes that polymorphisms are noise caused by random genetic drift and conserved regions are the result of strong negative selection indicating functionally relevant parts. Compared to purely selec-

tion-driven evolution, the neutral theory makes several testable, quantitative predictions (Kimura 1991; Takahata 1996). For example, the neutral theory predicts that the probability of fixation for a neutral mutation is equal to its current frequency in the population and hence that the fate of a mutation depends on random genetic drift. In this case, it can be shown that the rate of neutral mutation is equal to the rate of substitution. Furthermore, the genetic diversity in a population, i.e. the average number of differences between two randomly sampled sequences, is proportional to $N\mu$ where N is the population size and μ is the mutation rate per generation. The neutral theory also provides a rationale why phylogenetic trees can be inferred reliably from genetic data: as the neutral theory predicts that sequences diverge over time by accumulating random mutations at a rather constant pace, the time of divergence of two sequences is proportional to their genetic distance and the evolutionary tree can be recovered from these distances. The constancy of rates is also referred to as the molecular clock. The molecular clock has been useful for dating evolutionary events in the past, even though its estimates suffer from a high variance and confounding effects such as varying population sizes (Bromham and Penny 2003). But the neutral theory has also lead to the development of other quantitative sequence analysis methods besides phylogenetic tree inference. For example, tests for neutral evolution allow detecting deviations from the neutral model and provide evidence that selection played a role (Bamshad and Wooding 2003).

3.1.1 Selection and Drift in Sequence Analysis

Selection and random genetic drift are two very different forces that shape genetic sequences. Drift is caused by the ancestral relationship of the sampled sequences and the mutation process, while selection shapes the genetic sequence mainly depending on the environment and the genotype-phenotype relationship. Sequence analysis often tries to elucidate the cause of polymorphisms, for example the ancestral relationship or the genotype-phenotype relationship. If drift dominates, the ancestral relationship and the mutation process can be estimated, for example by phylogenetic inference if the ancestral relationship can be described by a tree. On the other hand, association studies try to find polymorphisms that cause a disease and explain part of the genotype-phenotype relationship. In essence, association studies look for genetic regions that discriminate between diseased and healthy individuals, as these genetic regions are more likely to cause the disease.

The analysis is complicated considerably if both, selection and drift play a role in molecular evolution. In this case, sequence similarity is not only caused by common ancestry, but also by convergent evolution due to a similar selection pressure. It is difficult and often impossible to discern these effects in an analysis (Bamshad and Wooding 2003). For example, consider estimating a phylogenetic tree if selection dominates the evolution of the genetic sequence. The phylogenetic tree then may not represent the ancestral relationship anymore, but rather the similarity of the selective environment associated with the

sequences – and the analysis becomes biased. It is more frequent that the analysis of selection is complicated by common ancestry, as common ancestry and drift always plays an important role in sequence evolution. Common ancestry essentially leads to linkage between sites and can confound the genotype-phenotype relationship in the search for disease markers. If all individuals that suffer from a genetic disease are closely related, their genetic makeup is very similar because of their common ancestry and does not contain much information on the cause of the disease. Hence, common ancestry is usually taken into account in the analysis of selection if individuals in the study differ widely in their degree of kinship. An example is linkage analysis, which focuses on samples of individuals that are related by a known pedigree. Interestingly, recombination also plays a major role in the analysis of disease genes, as it breaks up linkage and restricts the effect of common ancestry to small regions of genetic material. Recombination therefore helps to localize disease genes more accurately.

The subject of this thesis is the analysis of molecular sequence data regarding recombination events. Recombination events do not cause point mutations and therefore do not leave direct evidence in the observed sequence data. Consequently, recombination events solely affect the ancestry of the sequences and we can only derive information on recombination events from neutrally evolving sites. Selection interferes with recombination analysis to a much larger extent than it does with phylogenetic inference. As recombination breaks up linkage, recombination analysis inherently depends on evaluating sequence similarity only for a small region in the sequences and not for full length sequences. The probability that a small region in the sequences shows similarity due to selection is much higher than for full length sequences – also because functionally related and jointly selected sites are often close by. Strong selection pressures may actually lead to patterns similar to that of recombination and lead to wrong results. For example, a strongly selected region can be similar in otherwise unrelated sequences and result in a pattern similar to recombination or gene conversion. Hence, the method developed later in this thesis assumes that mutations are neutral and selection does not occur. As recombination only transfers existing mutations between sequences, it is difficult enough to infer recombination events based on sequence data without the effect of selection.

3.2 Models of Nucleotide Substitution and Mutation

Duplicating a genetic sequence always bears a chance that the copy introduces one or several mutations and that the copy differs from the original sequence. These mutations are then subject to selection: we can only observe mutations that are non-lethal or not under strong negative selection. However, the copy process itself may introduce mutations by pure chance and at any position in the sequence, as predicted by the neutral theory (see section 3.1). The models introduced in the following are not only used for the nucleotide substitution process between species, but also for the mutation process within a population. Both, substitution and mutation process are very similar if the sequences evolve ac-

cording to the neutral theory. As the models in the literature are usually referred to as substitution models, the term substitution is used in the following even if the mutation process is more relevant for this thesis.

As the genetic sequence defines the biochemical composition of an organism, it also encodes the machinery for copying the genetic sequence. Consequently, the copy process is a memoryless process that does not depend on how the genetic sequence evolved so far, but only on the current state. Hence, the copy process can be described by a time-homogeneous Markov chain, at least if the environment does not change its effect on the copy process. In other words, each genetic sequence defines a separate probability distribution over the sequence space for the copies it can create. As the state space of such a Markov chain is too large, it is typically assumed that each sequence induces the same probability distribution and each nucleotide changes according to the same Markov chain. In other words, the changes of each nucleotide in the genetic sequence are described by an independent, but identically distributed Markov chain with the state space $\{a,g,c,t\}$. It is fairly simple to extend this model and allow for example heterogeneous mutation rates at different genome positions. Some of these extensions are discussed later in this section.

In the following, the substitution models are presented as time discrete Markov chains. More details regarding the substitution models and an introduction to continuous time Markov chains are presented in (Ewens and Grant 2001). A more comprehensive and more practical introduction is given in (Li 1997). Both books also discuss amino-acid substitution models. As amino-acid substitution models are also implemented with Markov chains and share a lot of similarity with nucleotide substitution models, they are not presented here. Finally, these books also describe extensions that allow for a different substitution rate at different positions in the genome.

Let p_{xx} be the probability that a nucleotide x remains unchanged after one copy operation and analogously define p_{xy} as the probability of a change from nucleotide x to nucleotide y after one copy operation. The discrete-time Markov chain is then best represented as a matrix, where the transition probability from one state to all other states is given as a row in the matrix:

$$\mathbf{P} = \begin{bmatrix} p_{aa} & p_{ag} & p_{ac} & p_{at} \\ p_{ga} & p_{gg} & p_{gc} & p_{gt} \\ p_{ca} & p_{cg} & p_{cc} & p_{ct} \\ p_{ta} & p_{tg} & p_{tc} & p_{tt} \end{bmatrix}. \tag{3.1}$$

As each row describes the probability of a transition from one nucleotide to all others, the rows in the matrix sum to 1. The columns sums amount to the probability of a transition to a specific nucleotide. Now, define $\mathbf{x}=(x_a,x_c,x_g,x_t)$ as a vector of probabilities, where x_a is the probability for nucleotide a. Then $x_a+x_c+x_g+x_t=1$ and the resulting probabilities after a transition are computed as $\mathbf{xP}$. For example, if the frequency of nucleotides in a sequence is described by a vector $\mathbf{x}$, the expected frequency of nucleotides in the sequence after one transition is $\mathbf{xP}$.

All substitution models introduced in the following and almost all substitution models in literature are finite, aperiodic and irreducible Markov chains. As a consequence, they have a stationary distribution and are time-reversible. The stationary distribution $\boldsymbol{\pi}$ is the eigenvector of $\mathbf{P}$ with eigenvalue 1 and therefore does not change after a transition: $\boldsymbol{\pi}\mathbf{P}=\boldsymbol{\pi}$. For the Markov chains considered here, any probability vector $\mathbf{x}$ converges towards $\boldsymbol{\pi}$ the more transitions it undergoes: $\mathbf{x}\mathbf{P}^n \rightarrow \boldsymbol{\pi}$ as n goes to infinity. Thus, the nucleotide frequencies of a sequence become more and more similar to $\boldsymbol{\pi}$ as the sequence evolves. If a Markov chain is time reversible, it is impossible to infer whether a sequence of state changes was recorded forward or backward in time. To be more specific, taking into account the prior probability for observing nucleotides x and y, the probability of a transition from x to y is equal to the probability of a transition from y to x. In mathematical terms, the probability matrix therefore has to satisfy the detailed balance equation $\pi_x p_{xy}=\pi_y p_{yx}$ for all states x and y. There is an important detail if the substitution process is simulated: the starting sequence should be drawn from the stationary distribution of the substitution process. If the composition of the starting sequence deviates from the stationary distribution, every transition renders the sequences closer to the stationary distribution on average, and the direction of time can be inferred.

The simplest model of nucleotide substitution, the Jukes-Cantor (JC) model (Jukes and Cantor 1969), assumes that there is a single parameter α, the probability of substituting a nucleotide with any other nucleotide. The JC model is described by the following transition matrix

$$\begin{bmatrix} 1-3\alpha & \alpha & \alpha & \alpha \\ \alpha & 1-3\alpha & \alpha & \alpha \\ \alpha & \alpha & 1-3\alpha & \alpha \\ \alpha & \alpha & \alpha & 1-3\alpha \end{bmatrix} \tag{3.2}$$

The stationary distribution of the JC model is (0.25, 0.25, 0.25, 0.25) and assumes that each nucleotide occurs with the same probability. It is usually necessary to normalize a substitution model such that the probability of mutation is equal to the mutation rate μ per nucleotide per genome. The probability of mutation is 3α for the JC model, regardless of the current state and it is sufficient to set $\alpha=\mu/3$ to normalize the JC model. Obviously, normalization eliminates one parameter and the JC model reduces to a parameter-free model if the mutation rate is given.

To obtain a more realistic model, Kimura proposed the Kimura-two-parameter (K2P) model (Kimura 1980) that introduces different transition probabilities α and β for *transitions* and *transversions*, respectively. *Transversions* are changes between states in $\{a,g\}$ and states in $\{c,t\}$, where $\{a,g\}$ are purines and $\{c,t\}$ are pyrimidines. *Transitions* refer to changes within the categories of purines and pyrimidines. The transition matrix is

$$\begin{bmatrix} 1-\alpha-2\beta & \alpha & \beta & \beta \\ \alpha & 1-\alpha-2\beta & \beta & \beta \\ \beta & \beta & 1-\alpha-2\beta & \alpha \\ \beta & \beta & \alpha & 1-\alpha-2\beta \end{bmatrix}. \tag{3.3}$$

The stationary distribution of the K2P model also assumes that each nucleotide occurs with the same probability. Again, normalizing the substitution model to a given mutation rate μ is simple. Define the transition-transversion ratio as $\kappa=\alpha/\beta$ and observe that the rate of mutation is $\alpha+2\beta=\kappa\beta+2\beta$. If we set $\beta=\mu/(\kappa+2)$ and $\alpha=\kappa\mu/(\kappa+2)$, the K2P model has the same transition-transversion rate and introduces a mutation with probability μ.

There are several other substitution models introducing additional parameters. Some allow for a stationary distribution that is fully parameterized, while others have additional parameters to distinguish between different types of state transitions. The general time reversible model has ten parameters and as its name suggests is the most general model that is still time reversible. Other substitution models capture the influence of neighboring nucleotides and have a much larger state space. Codon models, for example, use nucleotide triplets as states in the model, leading to a total of 64 states. As the number of states and the number of free parameters grows, it becomes more difficult and unreliable to estimate all model parameters from sequence data.

The extensions described so far either pursue a more detailed description of the Markov chain or relax the independence assumption among nucleotides. But the assumption that every nucleotide evolves identically to all others is often violated, too. As it is infeasible to model each position with a different Markov chain, it is usually assumed that one Markov chain describes the evolution of all positions well enough. However, every position i could have a different substitution rate $\lambda_i\mu$, where λ_i modulates the substitution rate for position i. The mean of λ_i is usually set to one, such that μ represents the average substitution rate. A value λ_i smaller or greater than one may then represent a functional constraint or a mutation hotspot at position i, respectively. A common assumption is that the λ_i are drawn from a gamma distribution with a mean of one (Yang 1996). Setting the mean to one eliminates one parameter from the gamma distribution and leaves a single shape parameter γ. The shape parameter γ represents the amount of rate variation. A high value of γ corresponds to a low amount of rate variation, and for $\gamma=\infty$ all sites evolve according to the same rate. Again, the model does not specify which positions evolve according to some rate λ, but only that the distribution of the rates obeys the gamma distribution.

3.3 The Ancestry of a Sample of Genes

The following section introduces several concepts and models of population genetics that describe the ancestry of a sample of genes, the so-called genealogy. It is important to model the genealogy with recombination, as recombination only affects the genealogy and can only be inferred from the genealogy. We first introduce some terminology and then describe several population genetic models, such as the Wright-Fisher model. Finally, we introduce the coalescent with recombination and the Ancestral Recombination Graph (ARG) as an accurate model of the genealogy with recombination.

The term “gene” in this section only refers to the object of evolution; it is equivalent to a chromosome or any other continuous genetic sequence. In particular, individuals in the

population are reduced to their genetic information. The term individual either corresponds to a single gene if the genome is haploid or to a pair of genes if the genome is diploid. Population genetics usually describes the evolution of individuals in a homogeneous population and estimates population parameters such as population growth or migration rates. Conversely, phylogenetics classically studies evolution at a coarse scale and addresses the ancestral relationship between species. Hence, the terms phylogeny and genealogy refer to species evolution and gene evolution, respectively. Both terms are not consistently distinguished in literature, though, as methods for phylogenetic inference also work for data sampled within a population.

Population genetic models of evolution address a diverse set of questions regarding evolution. They were first used to study evolution theoretically and to derive expectations for parameters, such as sequence diversity or number of polymorphic sites. These parameters are also used to design statistical tests, for example to test for neutral evolution (Simonsen, Churchill et al. 1995). As population genetic models are quantitative descriptions of the process of evolution, they are also often used to simulate the ancestry of sequence datasets. In contrast to biological data, the parameters of evolution and in particular the ancestry that generated the sequence dataset are known for simulated sequences. This is especially important for validation in our case, as biological datasets with a known history of recombination events do not exist. Inference is also possible in population genetics, but presents a much more involved problem. There are several programs for estimating population genetic parameters, such as population growth or migration rate from sequence data. Finally, population genetics introduces an accurate model for the ancestry with recombination, the ARG. In essence, recombination analysis infers some parameters of the ARG, for example, the recombination rate or the position of recombination breakpoints. The mutation process itself is usually not affected by recombination, and the ARG contains the complete information on the recombination events. The ARG also provides a platform for a more thorough analysis of the effect of recombination and helps to classify recombination events into different categories of complexity.

There are several good books that describe the Wright-Fisher model and the coalescent. Most of this section is based on these books. Felsenstein (Felsenstein 2004) gives a short and illustrative introduction to these models. The book by Hein and coworkers (Hein, Schierup et al. 2005) presents many more details and introduces applications of the Wright-Fisher model and the coalescent along with several real-world examples. Finally, Tavaré (Tavaré 2004) gives a detailed mathematical treatment of these models.

3.3.1 The Wright-Fisher Model

A simple, but influential and instructive probabilistic model of genealogies is the Wright-Fisher model. The Wright-Fisher model builds genealogies forward in time, starting in the past and progressing towards the present. Each generation is non-overlapping and consists of N haploid individuals or genes. Genes of generation t+1 are created by

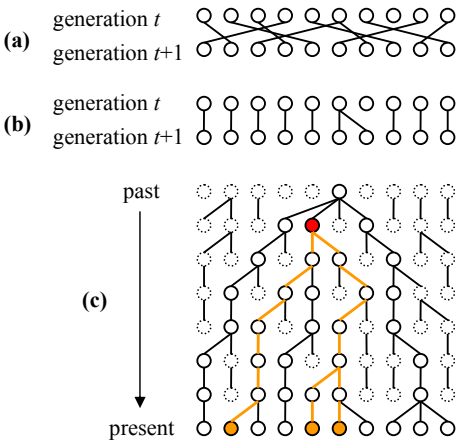


Figure 3.1: Realization of the Wright-Fisher model for a population of N=10 genes. Genes are represented by circles. **(a)** Genes in generation t+1 choose their parents randomly from the genes in generation t. **(b)** As the ordering of the genes is arbitrary, they can be reordered to untangle the lines of descent between generations. Some genes do not give rise to progeny and their lineage dies out. **(c)** Genes that are not ancestral to the present generation are represented by dashed circles. As there is no information about these genes in the present generation, they are irrelevant for the analysis of the present generation. A sample of three genes and their descent have been marked in orange. The MRCA of the three genes is shown in red. All sets of genes of the present generation find their MRCA within the displayed eight generations.

randomly choosing a gene of generation t as depicted in Figure 3.1. As an example, the first gene in generation t+1 chooses each gene in generation t with probability $1/N$. A natural consequence of randomly choosing parents is that some lineages die out, while others duplicate by pure chance. Going back in time, lineages coalesce when genes choose the same parent, and the genealogy can be described by a tree as in Figure 3.1(c). Hence, the lineages of any sample of genes eventually go back to a single gene, the most recent common ancestor (MRCA) of the sample. The MRCA is of particular interest: the sample does not contain any information on evolutionary events that happened before the time of the MRCA. Conversely, only a single gene of the present generation is the ancestor of all genes in some future generation. This is also the cause for random genetic drift: a variant of a gene becomes fixed in the population purely by chance effects.

The Wright-Fisher model affords a simple scheme for simulating random genealogies. It is usually assumed that only a small sample of k genes is of interest and not the whole population, and that all sampled genes are drawn from the same time point. Starting in the past, successive generations of N genes are simulated forward in time until all genes in

the sample are derived from a single MRCA within the simulated time span. This ensures that every gene evolved from a single ancestor. The ancestry of the sample can then be traced backwards in time. A particularly attractive feature of the Wright-Fisher model is that it easily extends to various scenarios, such as selection, migration and recombination. In these cases, sequence simulation is still straightforward, but inference becomes difficult or even impossible.

The Wright-Fisher model as described above is based on several assumptions. Fortunately, many assumptions do not affect its generality, as deviations can often be modeled by adjusting the population size N. To be more specific, the effective population size N_e of a real world population refers to the size N of a population in the Wright-Fisher model that in some sense approximates the real world population best (Hein, Schierup et al. 2005). In other words, the Wright-Fisher model with population size N_e is a good approximation of the real world population. Obviously, the Wright-Fisher model with population size N has an effective population size of N_e=N. In detail, the assumptions of the Wright-Fisher model are (Hein, Schierup et al. 2005):

Discrete and non-overlapping generations

All individuals of a generation exist exactly for the same duration and die at the time point when the next generation starts to exist. Even though this assumption is unrealistic, it turns out that models with overlapping generations produce very similar genealogies if the effective population size is adjusted accordingly.

Constant population size

Each generation is composed of exactly N individuals or genes. An extension of the Wright-Fisher model to varying population sizes is obvious: generation t has N_t individuals that are sampled randomly from the N_{t-1} individuals of generation t-1. Varying population sizes can clearly affect the generated genealogies and a simple adjustment of the effective population size does not suffice. For example, if the population grows exponentially, there are more genes in each consecutive generation and the probability that a gene becomes extinct by random sampling is much smaller. It is easy to extend the coalescent to varying population sizes both for simulation and inference.

Haploid individuals and no male and female subpopulation

The haploid and the diploid Wright-Fisher model do not differ much: a diploid individual merely combines two genes. There is actually no difference for the structure of the genealogy if selfing is allowed and there are no male and female subpopulations. The two genes of a diploid individual each pick a random parent gene from the previous generation as before, ignoring the pairing of genes introduced by diploidy. The only difference is that a diploid model has two genes per individual and 2N genes in the whole population. Hence, the diploid model is obtained from the haploid model by substituting N by

$2N$ in all formulas. If there are male and female subpopulations as in the human population, each individual picks one random gene from the male subpopulation and one random gene from the female subpopulation. The resulting genealogies can be modeled by an effective population size that is lower than the actual population size.

No migration and no geographical or social structure

Individuals in a population are often more likely to mate with geographically close individuals or individuals of a similar social status. There are several approaches to this scenario. A classical approach is the two island model, where the progeny of an individual usually stays on the same island. The genes in the Wright-Fisher model can therefore be partitioned into two groups. Genes of each group usually pick their parents from the same group, unless there is a migration event.

No fitness difference among genes

Apparently, the Wright-Fisher model assumes that each gene has the same chance to reproduce, irrespective of the genetic sequence. It is straightforward to extend the Wright-Fisher model to selection and introduce fitness as a function of the genetic sequence. Let f_i be the fitness of the i-th gene in the parental generation and assume that f_i is normalized such that the f_i's sum to one. We can then model the effect of selection by choosing each gene i with probability f_i as a parent. After the parents of all genes in the current generation have been determined, the fitness f_i of each gene in the current generation is computed and the procedure repeats. Even though it is simple to simulate the Wright-Fisher model with selection, there is an important difference to the case without selection: mutations are not neutral anymore, and the genetic sequences have to be simulated together with the ancestry. Hence, selection slows down sequence simulation and makes inference very difficult.

No recombination

Recombination deeply changes the structure of the genealogy as each recombinant is derived from two parental genes in the simplest case. The probability of recombination per gene per generation is referred to as r in the following. In other words, every gene in the current generation is derived by recombination with probability r and then picks two distinct parents randomly from the previous generation in that case. Otherwise the gene selects a single parent as before. The recombinant is usually derived from the parents by equal recombination as in Figure 2.6, without insertion and deletion. The recombination breakpoint is often assumed to occur uniformly distributed along the genetic sequence.

3.3.2 The Coalescent Without Recombination

Even though the Wright-Fisher model is very intuitive for simulating sequences, it is quite difficult to obtain analytical results and estimate parameters of the model such as

the mutation rate μ. In 1982 Kingman (Kingman 1982) introduced the coalescent, a continuous-time approximation of the Wright-Fisher model that only has a single parameter: the effective population size N_e. As this section is based on the standard Wright-Fisher model where N_e=N and for notational convenience, N_e is referred to as N in the following. The coalescent is introduced for a haploid population, but the diploid coalescent is just as common in literature. If the population is diploid, N must be substituted by $2N$ in all formulas to account for the fact that there are twice as many genes than in a haploid population. The coalescent is based on the assumption that only a small sample of k genes is of interest and that $k(k\text{-}1)$ is small compared to N. Kingman showed that the coalescent is a robust approximation of many different discrete-time population genetic models. It is often sufficient to calibrate the effective population size and hence the time-scaling of the coalescent to approximate different discrete-time models. As the analysis of Kingman is mathematically involved, only a simple motivation for the coalescent is given in the following.

In contrast to the Wright-Fisher model, the coalescent studies the ancestry of a sample of k genes looking back in time and observes how lineages coalesce and reduce in number. The idea is to approximate the time until the first coalescent event, as it reduces the number of lineages by one. To be more specific, let T_k denote the time measured in number of generations until a coalescent event reduces the number of lineages from k to k-1. In the Wright-Fisher model, the probability that two particular lineages coalesce in the parental generation is $1/N$. Hence, the probability that two particular lineages coalesce exactly j generations back in time is

$$\Pr(T_2 = j) = \left(1-\frac{1}{N}\right)^{j-1}\frac{1}{N} \tag{3.4}$$

and T_2 is geometrically distributed with parameter $1/N$. As $(1+1/x)^x \approx e$ and therefore $(1\text{-}1/x)^x \approx 1/e$ for large x, T_2 can be approximated by an exponential distribution if N is large:

$$\Pr(T_2 > j) = \left(1-\frac{1}{N}\right)^{j} = \left(1-\frac{1}{N}\right)^{N\frac{1}{N}j} \approx e^{-\frac{1}{N}j}. \tag{3.5}$$

A similar expression can be derived for T_k using an additional approximation. From the Wright-Fisher model it follows that the probability that all k genes in the sample choose different parents is

$$\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)\ldots\left(1-\frac{k-1}{N}\right) \tag{3.6}$$

as the first gene may choose any parent, the second gene may choose any of the remaining N-1 parents and so on. If we multiply all terms out, we get

$$\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)\ldots\left(1-\frac{k-1}{N}\right) = 1-\frac{(1+2+\ldots+(k-1))}{N}+O\left(\frac{1}{N^2}\right) = 1-\frac{k(k-1)}{2N}+O\left(\frac{1}{N^2}\right). \tag{3.7}$$

It is valid to ignore terms with a divisor of N^2 or larger as $k(k\text{-}1)$ is much smaller than N. In essence, the approximation assumes that there is not more than one coalescent event in

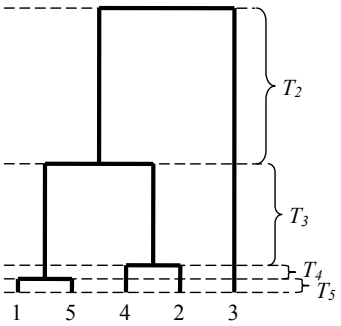


Figure 3.2: A coalescent tree as a random realization of the coalescent process for a sample of k=5 genes. The population size N is immaterial here, as it only determines the time scaling. Vertical branch lengths are proportional to time and all genes are sampled at the same time point in the present. Apparently, more time is spent waiting for the last few lineages to coalesce.

every generation. Given the probability that no coalescent event occurs in one generation, it is again easy to derive the probability that no coalescence event occurs for j generations

$$\Pr(T_k > j) = \left(1 - \frac{k(k-1)}{2N}\right)^j \approx e^{-\frac{k(k-1)}{2N} j}. \tag{3.8}$$

The approximation effectively substitutes the geometric distribution by an exponential distribution, which is reasonable if $k(k\text{-}1)$ is small compared to N. In other words, the waiting time until any pair of k lineages coalesce is exponentially distributed with parameter $k(k\text{-}1)/2N$ if time is measured in generations. It turns out that many mathematical expressions are simpler if we measure time in units of $2N$ generations. Define t as the time measured in units of $2N$ generations, such that $t=j/2N$. The waiting time T_k^c measured in $2N$ generations until any of k lineages coalesce is then exponentially distributed with parameter $k(k\text{-}1)$:

$$\Pr\left(\frac{T_k}{2N} > \frac{j}{2N}\right) = \Pr\left(T_k^c > t\right) \approx e^{-\frac{k(k-1)}{2N} 2Nt} = e^{-k(k-1)t}. \tag{3.9}$$

The formulation shows that the coalescent process is the same for populations of different sizes – only the time scaling changes.

As the coalescent only tracks lineages ancestral to the sample, it is much easier to obtain analytical results and perform inference. For example, a simple result of the coalescent process is that more and more time is spent waiting for the last few lineages to coalesce. The sum of the waiting times for k lineages to coalesce to a single lineage is

$$\mathrm{E}\left[\sum_{i=2}^{k} T_i\right] = \sum_{i=2}^{k} \mathrm{E}[T_i] = \sum_{i=2}^{k} \frac{2N}{i(i-1)} = 2N\left(1 - \frac{1}{k}\right). \tag{3.10}$$

A sample of k genes therefore needs on average less than $2N$ generations to find its MRCA. The expected time for the last two lineages to coalesce is N generations. In other words, approximately half of the time is spent waiting for the last two lineages to coalesce, a fact that is clearly visible in Figure 3.2.

Simulating genealogies with the coalescent is very efficient, as only the lineages of genes in the sample are traced backwards in time. Given the size of the sample k, the waiting time until the first coalescent event is drawn from an exponential distribution with parameter $k(k\text{-}1)$. Then, two random lineages are drawn from the k lineages of the sample and their lineage is coalesced. Finally, k is decreased by one and the process repeats until k is equal to 1. A random realization of the coalescent for k=5 is presented in Figure 3.2. In contrast to phylogenetic trees, coalescent trees are always rooted, have branch lengths proportional to time and usually assume that all genes are sampled at the same time point.

In summary, the coalescent essentially generates genealogies by drawing k-1 waiting times and then drawing a random tree topology. In other words, the coalescent not only separates the process generating the genealogy from the mutation process, but further splits the genealogical process into one that generates times of coalescence events and another one that generates tree topologies. Compared to the simulation process of the Wright-Fisher model, the coalescent is much more efficient as it only tracks the genealogy of the sample and never simulates events that do not affect the genealogy. However, the coalescent is not as flexible and intuitive as the Wright-Fisher model. Extensions to incorporate population structure, migration or selection are not trivial.

Particularly for inference, but also for simulation, it is important to incorporate the mutation process into the coalescent. In the simplest case, the mutation process is modeled with a constant rate of μ mutations per genome per generation. There are more complex models of the mutation process, but it is usually possible to derive an average mutation rate μ (see section 3.2). As time is scaled in units of $2N$ generations in the coalescent, it makes sense to measure the mutation rate in units of $2N$ generations, too. The coalescent with mutation therefore introduces a parameter θ=$2N\mu$ that measures the mutation rate per genome per unit of scaled time t. A simple result is that there are on average θ mutations per sequence from the root to a leaf of a random coalescent tree, as the expected height of the tree is $2N$ generations. For simulating sequences, the only relevant parameters are k and θ, but not N: using k as an input, the coalescent generates a random genealogy, where branch lengths are measured in $2N$ generations. But the time scaling is irrelevant as it only matters that there are on average θ mutations for any branch of length 1. Given the genealogy, the mutation process can then be simulated forward in time as in the Wright-Fisher model. Using a similar reasoning, it is only possible to infer the compound parameter θ, but not N if all sequences are sampled at the same time point and the mutation rate μ is unknown.

3.3.3 The Coalescent With Recombination

The Wright-Fisher model with recombination assumes that recombination events occur with a rate of r recombinations per gene per generation. If a gene is created by recombination, two parents are chosen from the ancestral generation and their genomes are combined. Looking back in time, a recombination increases the number of lineages by one in the ancestral generation. The probability that a recombination event and a coalescent event occur in the same generation is negligible if N is large and recombinations are not too frequent. Hence, recombination events and coalescent events are independent of each other and their rate can be studied separately.

Define T_R as the waiting time until one recombination event occurs in a single lineage. The probability that a recombination event occurs j generations back in time is

$$\Pr(T_R = j) = r(1-r)^{j-1} \tag{3.11}$$

and T_R is geometrically distributed with parameter r. Using the same approximation as in (3.5), T_R can be approximated by an exponential distribution if r is small:

$$\Pr(T_R > j) = (1-r)^j = (1-r)^{\frac{1}{r}rj} \approx e^{-rj} \tag{3.12}$$

In the following, time t is measured in units of $2N$ generations, such that $t=j/2N$. The waiting time T_R^c for a recombination event in a single lineage is distributed as

$$\Pr(T_R^c > t) \approx e^{-r2Nt} = e^{-\rho t} \tag{3.13}$$

where $\rho=2Nr$ is the scaled recombination rate. As recombination events occur in every lineage independently, the waiting time for a recombination in any of k lineages is approximately exponentially distributed with parameter $k\rho$. As shown before, the waiting time for a coalescent event is exponentially distributed with parameter $k(k\text{-}1)$. Hence, the waiting time until any of the two events occur is exponentially distributed with parameter $k\rho+k(k\text{-}1)$. The probability that the event is a recombination is

$$\frac{k\rho}{k\rho + k(k-1)} = \frac{\rho}{\rho + k - 1} \tag{3.14}$$

and the probability that it is a coalescent event is $(k\text{-}1)/(\rho+k\text{-}1)$. Based on these observations, Hudson (Hudson 1983) described a method for randomly drawing genealogies from the coalescent with recombination. As before, the genealogy is constructed starting in the present looking back in time:

1. while $k>1$
2. draw the waiting time till the next event from an exponential distribution with parameter $k\rho+k(k\text{-}1)$
3. with probability $(k\text{-}1)/(\rho+k\text{-}1)$ it is a coalescent event: pick two random lineages, coalesce them, decrease k by one and go to 1.
4. with probability $\rho/(\rho+k\text{-}1)$ it is an recombination event: pick a random lineage, split it in two lineages, increase k by one and go to 1.

As the rate of recombination is linear in k and the rate of coalescence is quadratic in k, all lineages coalesce eventually and the algorithm terminates. The genealogy of a sample subject to recombination is represented most accurately by the ancestral recombination graph (ARG), a concept that is described in more detail by Griffiths and Marjoram (Griffiths and Marjoram 1996; Griffiths and Marjoram 1997). Looking back in time, an ARG visualizes how lineages split up and coalesce, until they find their MRCA (see Figure 3.3).

The recombination breakpoints are often annotated in the ARG and add a lot of complexity. We usually assume that all loci in a sequence are represented by the interval [0,1] and recombination breakpoints are numbers in this interval. Looking forward in time, a recombination event takes the left part of one ancestral gene and the right part of the other ancestral gene to compose the gene in the next generation. Looking back in time, the situation is analogous: a recombination event distributes the ancestral material onto two genes in the parental generation. Ancestral material refers to all loci of a lineage that are relevant for the sample and is sometimes also annotated in the ARG (see Figure 3.3).

Recombination breakpoints and the corresponding ancestral material are important as they identify which loci of a lineage are relevant for the sample. If all loci of a lineage are irrelevant for the sample, the lineage does not contain any ancestral material and is irrelevant for the sample. Another interesting effect is that some coalescence events bring together lineages that do not have any overlapping ancestral material. These coalescence events are referred to as non-merging coalescence events in the following, as they do not merge ancestral material. Finally, we can also recover the ancestry for a single locus. For each locus only one of the two lineages created by recombination is relevant, as the ancestral material of the other lineage does not contain the locus. A recombination is therefore resolved to a simple connection between two lineages, and the ancestry for each locus is a coalescent tree. Alternatively, we can also follow the lineages of the sample back in time and decide for each recombination event which lineage to pursue. In summary, the set of all recombination breakpoints partitions the interval [0,1] into sets of loci, such that the ancestry of each partition is a local coalescent tree.

3.3.3.1 Hudson's Algorithm – Sampling ARGs Efficiently

Even though the algorithm described before is perfectly valid for sampling ARGs, Hudson (Hudson 1983) also introduced a much more efficient version. Hudson's algorithm is based on two observations: recombination events occurring in non-ancestral material may be irrelevant to the sample, and the MRCA can be different for different loci (Hein, Schierup et al. 2005).

Suppose that a recombination event occurs in a gene whose left part is not ancestral to the sample. If the recombination occurs in the non-ancestral part, one parent provides all ancestral material, while the other parent only provides the non-ancestral part left of the

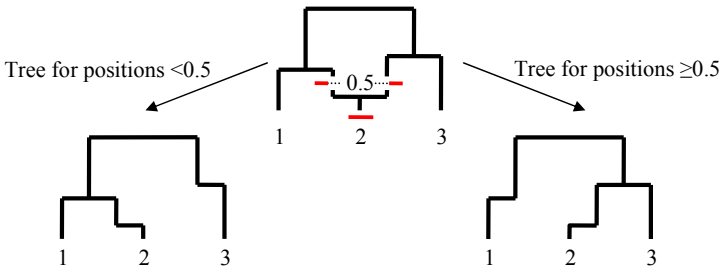


Figure 3.3: An ancestral recombination graph (ARG) for a sample of k=3 sequences. Sequence 2 evolved by recombination. The recombination occurred in the middle of the sequence, combining the left part of the ancestor of sequence 1 with the right part of the ancestor of sequence 3. The recombination breakpoint is annotated above the recombination event as a number in the interval [0,1]. The ancestral material of sequence 2 also visualizes the breakpoint and is shown as a fat red line, with dots indicating non-ancestral material. The coalescent tree for all positions <0.5 and ≥0.5 is also shown and describes the ancestry for a smaller region of the sequences. It is common to show the local coalescent trees with the same structure as they are embedded in the ARG, even though only the topology and the vertical branch lengths matter.

recombination breakpoint. Hence, one parent is irrelevant for the sample and the recombination event can be discarded from the ARG. There is an important special case: if non-ancestral material is trapped between ancestral material as shown in Figure 3.4(b), a recombination in the non-ancestral material is relevant for the sample, as it creates two lineages with ancestral material.

The other aspect that speeds up the simulation is that some parts of the sequence may find their MRCA long before the complete genes have found their MRCA (see Figure 3.4). It is simple to track the number of active lineages for each locus if the ancestral material is annotated in the ARG. As soon as there is only a single active lineage for a locus, the locus has found its MRCA. The tree describing the evolutionary history for the locus is known and the locus can be removed from the simulation. Hence, it is sufficient to obtain the coalescent tree at each locus – also termed local coalescent tree in the following – for sequence simulation.

3.3.3.2 Recombination as a Subtree-Prune-And-Regraft Operation

As noted above, an ARG defines a set of coalescent trees, where each tree is associated to some set of loci. Trees of adjacent loci are correlated and share much of the ancestry, as only some recombination nodes in the ARG are resolved differently between two adjacent loci. A recombination node has a simple effect on the local coalescent trees directly left and right of the recombination breakpoint: it removes an edge in the right tree and

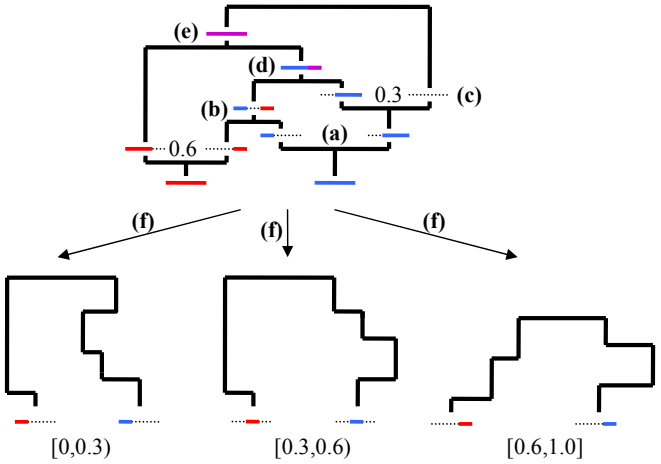


Figure 3.4: An ancestral recombination graph (ARG) for a sample of k=2 genes. The ancestral material of the left (right) sequence is shown in red (blue), and material that is ancestral to both sequences is shown in purple. Non-ancestral material is shown as a dotted line. **(a)** Looking back in time, a recombination event in the blue sequence distributes the ancestral material onto two lineages. The recombination breakpoint is located at position 0.3. **(b)** A coalescent event that creates a lineage with trapped non-ancestral material. **(c)** A recombination event that is irrelevant for the sample, as it leaves one lineage without ancestral material. **(d)** The right part of the sequence has already found its MRCA at this time point. **(e)** The complete sequence has found its MRCA. **(f)** The coalescent trees embedded in the ARG. The genetic region relevant for each tree is annotated below the trees. The left and the middle tree are identical, as the recombination in (a) is undone by the coalescent event in (d). Therefore, all loci in [0,0.6) find their MRCA at the same time point.

reattaches the subtree below the edge to some other branch to arrive at the left tree. A recombination can therefore also be described as a subtree-prune-and-regraft (SPR) operation on a coalescent tree (see Figure 3.5).

Not all SPR operations lead to an effect equivalent to a recombination event, as an SPR operation may contradict the time-ordering of coalescent events (Song and Hein 2003; Song 2006). Each coalescent event in the local tree occurs at a specific time point, and even more importantly, time introduces a relative ordering of the coalescent events. If the ordering of the coalescent events after an SPR operation is different from the ordering before, the SPR operation does not represent a recombination event (see Figure 3.6). Hence, an SPR operation corresponding to a recombination event must ensure that the relative order of coalescent events is retained. Conversely, inferring recombination events

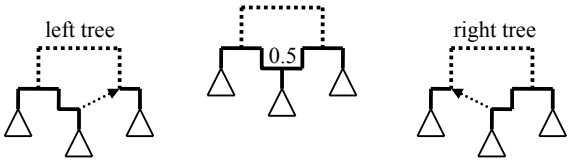


Figure 3.5: Recombination as an SPR operation. A recombination is displayed schematically in the center. Whole subtrees are shown as triangles and the rest of the ARG is shown as a dotted line. The left tree differs from the right tree by one SPR operation. The corresponding SPR operation is shown as a dotted arrow. The SPR operation cuts the branch at the base of the arrow and reconnects it to the branch at the tip of the arrow.

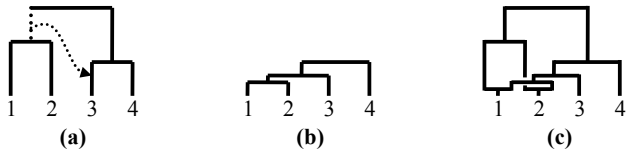


Figure 3.6: An SPR operation that contradicts the relative ordering of the CEs in the tree. **(a)** The tree before the SPR operation is shown. The dotted edge marks the edge to be removed and the arrow points to the destination in the tree. The coalescent events between the sequences {1,2} and {3,4} contradict this operation. **(b)** The tree resulting from the illegitimate SPR operation cannot be generated by a single recombination event from (a) as the coalescent event between 1 and 2 occurs at a different time point. **(c)** It is possible to construct an ARG that contains trees (a) and (b), but it requires two recombination events. The graph cannot be drawn without intersections and the recombination breakpoints are not annotated here. The only relevant observation is that sequences 1 and 2 coalesce before 3 and 4 for some locus, but much later for another locus.

always adds some time ordering to the coalescent events, as a recombination event can only occur after the lineages involved in the recombination exist.

SPR operations are also of interest for inference. Given the trees of two adjacent loci, a sequence of SPR operations that transform one tree into the other (and vice versa) can be interpreted as a set of recombination events with a recombination breakpoint between the loci. The minimum number of SPR operations needed to transform the trees of two adjacent loci is a lower bound on the number of recombination events at that locus in the ARG. Song (Song 2006) gives more details on SPR operations on ordered binary trees and derives bounds on the size of the neighborhood of a tree with respect to SPR operations. Hein and coworkers (Hein, Schierup et al. 2005) gives a nice example how the SPR

distance differs if the trees are unrooted, rooted, ordered or if they are coalescent trees and specify a time for each coalescent event.
It is remarkable that the idea of ARGs as a set of local trees also leads to a different algorithm for sampling ARGs. An ARG can be constructed starting at the leftmost locus and continuing to the right end of the sequence (Wiuf and Hein 1999). The algorithm starts with a random coalescent tree in the leftmost locus and adds more and more recombination events as it progresses towards the right. As the process has to keep track of all trees encountered, it is not a Markov process. It is also less efficient than Hudson's algorithm and is mainly of interest for a theoretical analysis of the coalescent with recombination.

3.3.3.3 Equivalent ARGs

The ARG is an accurate model for representing the evolution of sequences with recombination. But there is a certain problem when it comes to inference: it is impossible to distinguish between all ARGs using sequence data alone. The limitations of inference are presented in more detail in the following.
Reconstructing the ARG can only be successful if there is enough information in the sequence data. For example, the sequences should be long enough and the mutation process has to introduce enough polymorphisms. Assume in the following that the sequence data contain an infinite amount of information and that genetic distances can be estimated accurately. In other words, the time point when a locus last shared a common ancestor for two sequences is known. Another prerequisite is that each locus accumulates mutations independently of the other loci and correlated mutations do not occur.
As the distances are known, the coalescent tree for each locus can be inferred. The coalescent events in the local coalescent trees are also sufficient for simulating sequence data. Consequently, all other events do not change the sequence data and have to be inferred indirectly from the set of trees. A common characteristic of these events is that looking back in time they do not combine overlapping ancestral material from two or more lineages onto a single lineage. In other words, they are not "merging" ancestral material and hence we refer to these events as non-merging events in the following. The exact time point of a non-merging event is not specified by the local coalescent trees and therefore also cannot be inferred. There are only two types of non-merging events: recombination events in general, and some coalescent events. Looking forward in time a recombination event always combines the genetic material of both parents, but it does not merge ancestral material looking back in time. It is immaterial when the recombination event occurs, as loci accumulate mutations independently of each other. The only restriction is that both parental lineages have to exist before the recombination event and the recombinant lineage can only be involved in another event after the recombination event (see Figure 3.7, left ARG). Non-merging coalescent events satisfy a similar property. The non-merging coalescent event (b) in Figure 3.7 for example could occur at any time point between the surrounding events. This case also illustrates why it is important that each

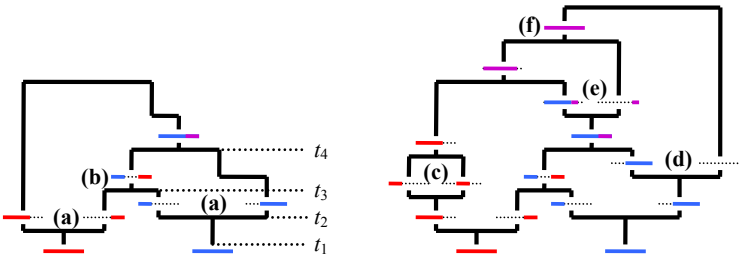


Figure 3.7: To the left, a modified version of the ARG of Figure 3.4 is shown, but many irrelevant events have been removed (see Figure 3.8 for a fully simplified ARG). Non-merging events are annotated and explained in more detail below. To the right, an equivalent ARG is shown that contains all types of irrelevant events. **(a)** Both recombination events are non-merging events and could occur at any time point between t_1 and t_3, or even up to t_4 if the coalescent event (b) is adjusted accordingly. **(b)** The time point of the non-merging coalescent event cannot be inferred from sequence data and could occur between t_2 and t_4. As both recombination events can also vary in time, the coalescent event could actually have occurred between t_1 and t_4, if the time points of the recombination events are adjusted accordingly. **(c)** Both, the recombination and the coalescent event are irrelevant. The recombination event creates two lineages which immediately coalesce again. **(d)** A recombination event in non-ancestral material is irrelevant if the non-ancestral material is not trapped between ancestral material. **(e)** A recombination event in ancestral material is irrelevant, as the ancestral material has already found its MRCA. **(f)** The coalescent event only merges ancestral material that has already found its MRCA. The loop created by recombination event (e) is therefore irrelevant.

locus evolves independently and that there are no correlated mutations. Correlated mutations could link the evolution of the left and right part of the sequence and the time point of the non-merging coalescent event would matter. But if correlated mutations do not occur, the time point of a non-merging coalescent event cannot be inferred.

Obviously, it is possible that the admissible time interval for a non-merging event depends on other non-merging events. In this case, the admissible time interval can be extended as the other non-merging events move within their admissible time interval. Even more interestingly, some non-merging events can cancel each other. The net effect of a recombination and coalescent event in succession is the same as if no event happened, as depicted in Figure 3.7 (a). Other events are also irrelevant and can be completely eliminated from the ARG without changing its representation as a set of coalescent trees (see Figure 3.6 and Figure 3.7).

In summary, many ARGs are equivalent in the sense that they lead to the same distribution of sequences. The ARG as a model of ancestry can be misleading for inference, as it

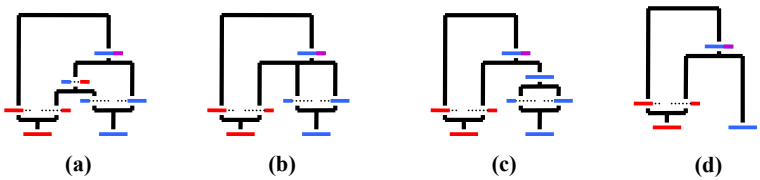


Figure 3.8: The ARG shown in Figure 3.7 can be further simplified. **(a)** The ARG in the left panel of Figure 3.7. **(b)** The non-merging coalescent event can be moved into the past, until a trifurcation occurs. **(c)** The trifurcation actually contains one non-merging coalescent event that can be moved towards the present: the coalescent event merging the lineages of the blue sequence. **(d)** Eliminating the loop in the blue lineages results in the fully simplified version of the ARG. It is not possible to further reduce this ARG, as every coalescent event is a merging event.

is only possible to infer the set of coalescent trees from sequence data but not always the ARG. Recombination events can only be inferred as SPR operations between coalescent trees of adjacent loci and are not explicitly represented by the set of coalescent trees. As the ARG is the only model that explicitly and accurately visualizes all events, the ARG is still the most appropriate model for describing evolutionary scenarios if all irrelevant events in the ARG are removed.

3.3.3.4 A Classification of Recombination Events

As recombination events can only be inferred indirectly from adjacent coalescent trees, it is possible to classify recombination events regarding the change that they cause in adjacent trees. This is important, as methods for the analysis of recombination often can only detect recombination events that change the unrooted topology of adjacent trees.

Looking back in time, a recombination event distributes the ancestral material of a lineage onto two lineages. Eventually, the ancestral material and the lineages are merged again onto a single lineage. As a result, each recombination event generates a loop in the ARG (marked red in Figure 3.9 and Figure 3.10). We can distinguish three types of recombination events depending on the number of lineages that coalesce to the loop created by a recombination event (Wiuf, Christensen et al. 2001; Hein, Schierup et al. 2005). Recombination events of type 1 occur if no lineage coalesces with the loop. In this case, the two lineages created by the recombination immediately coalesce again and the recombination event cannot be detected. Recombination events of type 2 occur if one lineage coalesces with the recombination loop or if there are three or fewer ancestral lineages at the time of the recombination event. Recombinations of type 2 result in a change of the branch lengths but do not change the unrooted topology (see Figure 3.9). The reasoning is simple: if only one lineage coalesces, the topology remains the same and only the branch

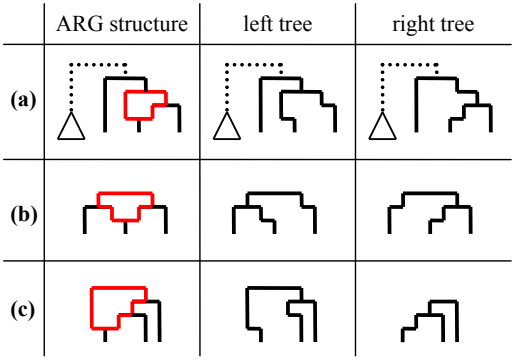


Figure 3.9: Recombination events of type 2 do not lead to a change in unrooted topology. The left and right tree is shown for comparison. **(a)** If the recombination loop only coalesces with a single lineage, the unrooted and rooted topologies do not change. It does not matter how many lineages exist at the time point of recombination, and this is visualized by the attached subtree. **(b),(c)** If the recombination loop coalesces with two lineages and there are three or fewer lineages at the time of the recombination event, the rooted topology changes, but the unrooted topology does not.

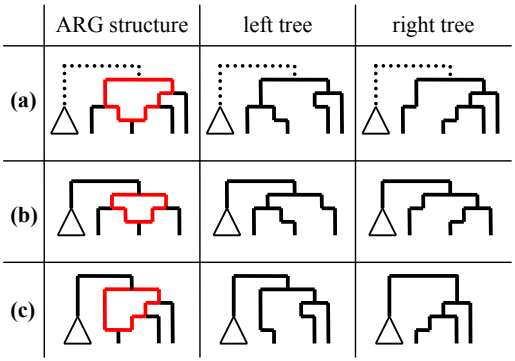


Figure 3.10: Recombination events of type 3 lead to a change in the unrooted topology. **(a)** The recombination event always changes the unrooted topology if more than two lineages coalesce to the loop. It does not matter how many other lineages exist, as the recombination structure already requires more than three existing lineages. **(b),(c)** These are the same events as in Figure 3.9, but an additional lineage exists at the time of the recombination event. As a result, the unrooted topology changes, too.

lengths change. If there are three or less ancestral lineages at the time of the recombination event, the unrooted topology cannot change as there is only one possible unrooted topology for three lineages. Finally, all other recombination events are of type 3 and change the unrooted topology (see Figure 3.10). This only occurs if two or more lineages coalesce with the loop and if there are more than three lineages at the time of the recombination event. The authors also give expectations for each type of event in randomly sampled ARGs. In essence, the more sequences are sampled, the higher the fraction of recombination events of type 3 compared to recombination events of type 1 or 2.

It is actually easier to understand this classification based on the representation of recombination events as an SPR operation between trees. If an SPR operation results in a change of the unrooted topology, the SPR operation has to connect two branches that are separated by at least two coalescent events in the unrooted tree (see Figure 3.11). If the tree is rooted and the SPR operations work on rooted trees, the condition is almost the same. Just as before, an SPR operation changes the rooted topology if and only if it connects two branches that are separated by at least two coalescent events in the rooted tree. However, the root node in the rooted tree counts as a coalescent event, too. Consequently, there is only one possible scenario for an SPR operation that changes the rooted topology, but leaves the unrooted topology unchanged: the SPR operation must connect two branches separated by one coalescent event and the root node (see Figure 3.11).

The original classification by Wiuf and coworkers (Wiuf, Christensen et al. 2001) refers to a single recombination event in an ARG and does not – and also does not have to – consider the effect of non-merging coalescent events. It is possible to construct scenarios that are wrongly classified by the description based on recombination loops if there is more than one recombination event (see Figure 3.12). The classification introduced above can be extended for these cases: in essence, non-merging coalescent events are not represented in the local trees and therefore should not be counted as coalescent events in the recombination loop.

3.4 Implications for the Analysis of Recombination

This chapter introduced the ARG as a model for the ancestry with recombination. The ancestry is particularly relevant as recombination events do not leave any direct evidence in the sequence data and have to be extracted from the ancestry itself. However, we can only hope to infer the ancestry if the majority of mutations are neutral. In fact, neutral mutations are much more important for recombination analysis than for phylogenetic tree inference, as recombination breaks up linkage and only small segments of the alignment can be used to infer parts of the ancestry.

Furthermore, there are several other restrictions and limitations for the analysis of recombination. The ARG is an accurate description of the ancestry with recombination and provides complete information on the time point and other properties of all recombination events. Unfortunately, it is impossible to reconstruct the ARG from sequence data alone.

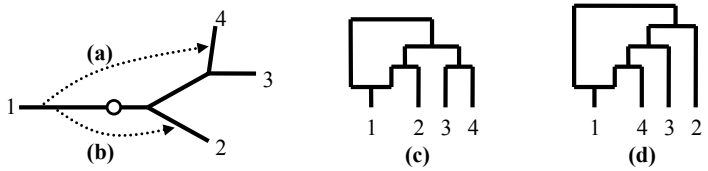


Figure 3.11: SPR operations that change the topology of the tree. **(a)** The SPR operation connects branches that are separated by two coalescent events and leads to a change in the unrooted topology. **(b)** The SPR operation only changes the branch lengths, but does not change the unrooted topology. If the tree is rooted at the white dot, the SPR operation (b) changes the rooted topology, but not the unrooted topology. **(c)** The ARG only containing the SPR operation (b). **(d)** The ARG only containing the SPR operation (a).

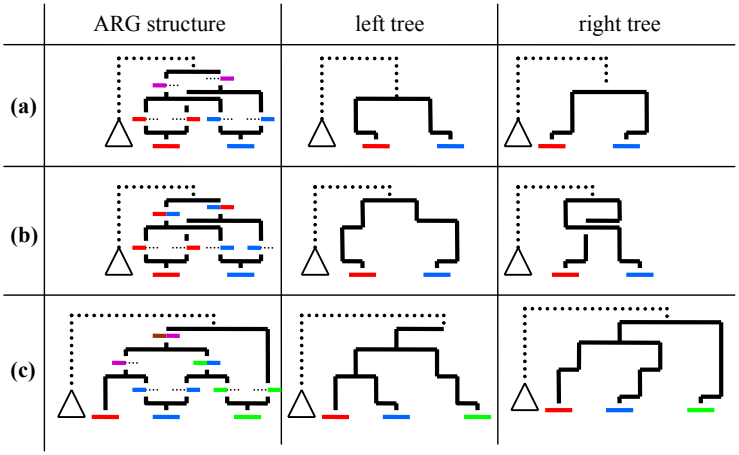


Figure 3.12: It is possible to construct ARGs that contradict the classification of recombination events based on the number of lineages merging to the recombination loop, particularly if the coalescent events inside the recombination are non-merging events. **(a)** Two recombination events of type 2 can cancel each other, such that both are recombination events of type 1 and do no leave any trace in the sequence data. The coalescent events for the left and right part are not exactly drawn on the same time point for visualization purposes. **(b)** The same overall effect can be achieved if both coalescent events in the recombination loops are non-merging. **(c)** It is also possible to construct a recombination event of type 3 that does not result in a change of unrooted topology.

In particular, the time point of non-merging events cannot be inferred, and some non-merging events can even be eliminated from the ARG without changing the probability distribution of the sequence data it generates. Consequently, several ARGs are equivalent regarding the sequence data they generate.

Alternatively, we can describe the ancestry with recombination as a set of coalescent trees. The set of trees is sufficient for simulating sequences and hence, we cannot hope to extract more information from sequence data alone. The set of trees is also uniquely defined if there is enough information in the sequence data. Hence, there is a one-to-one correspondence between the set of trees and sequence data in this case. Unfortunately, the set of coalescent trees does not directly provide information on non-merging events and in particular on recombination events. Recombination events have to be inferred indirectly as an SPR operation that transforms a local coalescent tree into the coalescent tree of a neighboring locus.

The representation as a set of trees also motivates a simple but very common strategy for recombination analysis: reconstruct evolutionary trees on sufficiently small regions of the genome and compare trees reconstructed from adjacent regions. Recombination is detected if adjacent trees differ significantly. Recombination events can subsequently be inferred as SPR operations. Such a strategy depends critically on many details. For example, the size of the region for tree inference has to be chosen very carefully. If the region is too small, we cannot infer the tree robustly. If it is too large, we risk inferring trees from recombining regions and end up with wrong trees and wrong results. Finally, it is also difficult to decide when two trees are significantly different.

This chapter also shows how to classify recombination events regarding the differences in the local coalescent trees. Recombination events of type 1 do not cause any change and cannot be inferred from sequence data. Recombination events of type 2 only cause a change in the rooted topology, while recombination events of type 3 also change the unrooted topology of the local coalescent tree. This classification is interesting as methods for recombination analysis sometimes can only detect changes in the unrooted topology. For example, it is notoriously difficult to root phylogenetic trees and we may only infer unrooted topologies for the strategy introduced above. As a result, the strategy cannot detect recombination events that only change the rooted topology and requires at least four sequences in the dataset.

4 Recombination Analysis

The following chapter serves as an introduction to recombination analysis and critically examines many existing methods. As recombination is a widespread biological process, there are many areas of research concerned with the analysis of recombinant sequences. For example, methods that search for disease markers or methods that construct genetic maps analyze recombinant sequences. However, the following chapter is restricted to the analysis of recombination in a strict sense: the goal is to infer recombination events in the ancestry of a set of sequences.

The task of recombination analysis is separated into several clearly defined steps that are often solved sequentially. The different steps in recombination analysis not only help to understand the problem of recombination analysis better, but also provide a system for classifying existing methods for recombination analysis. Similarly, assumptions that are common to several methods for recombination analysis are presented together with their implications. This is particularly useful as there are a rapidly increasing number of methods, which are often based on known concepts and assumptions. It is often possible to understand their advantages and disadvantages solely based on the assumptions inherent to the method. Finally, several methods for recombination analysis are presented and critically discussed.

4.1 Questions in Recombination Analysis

There are several types of questions that are relevant for the analysis of a sequence dataset regarding recombination. As the questions are presented by increasing complexity, the last question answers all previous questions, but is extremely difficult to answer. Therefore, it usually makes sense to approach a dataset step by step, begin with first question and check whether there is recombination in the dataset at all. Questions 2, 3 and 4 are mutually dependent and roughly have the same complexity, so that it does not matter if they are solved in a certain order or not.

1. **Is there recombination in the dataset?**

This is obviously the most fundamental question and is also called recombination detection. The result is of particular relevance. If there is no recombination in the dataset, many methods for sequence analysis can be used, for example methods to infer phylogenetic trees or to detect selected sequence positions. On the other hand, if there is recombination in the dataset, it is necessary to use special methods that work in the presence of recombination, as recombination can otherwise severely bias the results (Schierup and Hein 2000). It only makes sense to perform a more detailed analysis of recombination and target the other questions stated here, if there is recombination in the dataset. But it is also interesting from a biological standpoint to detect recombination in a dataset, as it provides evidence that the organism or virus is able to recombine. Methods for

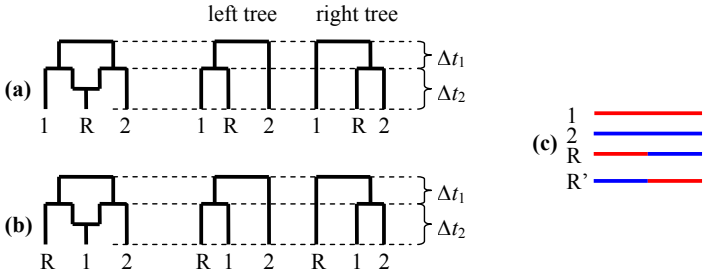


Figure 4.1: It is often difficult to determine from sequence data which sequence is the recombinant and which are the parental sequences. Two possible scenarios are presented and they are compared regarding the local trees they induce. **(a)** There are two parental sequences 1 and 2 and a recombinant R. **(b)** An alternative scenario where sequence 1 is the recombinant. The left trees are identical in both scenarios. Only the right trees differ and provide information about the correct scenario. The right trees are more dissimilar if Δt_1 is large and Δt_2 is small. The reason is simple: if Δt_1 is large and Δt_2 is small, the sequences R and 2 (1 and 2) in the upper (lower) right tree are more similar, and it is easier to keep both scenarios apart. **(c)** An alignment resulting from scenario (a) is shown schematically, where the color of the sequence indicates homology. If a hypothetical sequence R' is added, the recombination is symmetric and every sequence could be the recombinant.

recombination detection vary in the type of output given. Ideally, the method provides a *p*-value for the hypothesis that there is recombination in the dataset as a measure of confidence. Some methods only provide an arbitrarily scaled indicator for recombination, but it is often possible to transform the indicator into a *p*-value using the column permutation test (see section 4.2.4). As a variation of recombination detection, some methods even provide a quantitative measure for recombination and estimate the population scaled recombination rate ρ.

2. Where are the recombination breakpoints?

If the positions of the recombination breakpoints are known, the alignment can be split up into segments that evolved without recombination. Each segment can be analyzed by methods that do not account for recombination. In particular, it is possible to reconstruct phylogenetic trees for each segment and to analyze trees of neighboring segments regarding the SPR distance. Recombination breakpoints can also provide information on possible recombination hotspots or on the prevalent recombination mechanism in HIV-1. Some methods also give a list of recombination breakpoints and even provide the *p*-value for each breakpoint. However, there is a fundamental problem associated with detecting recombination breakpoints: every possible breakpoint in an alignment may involve a

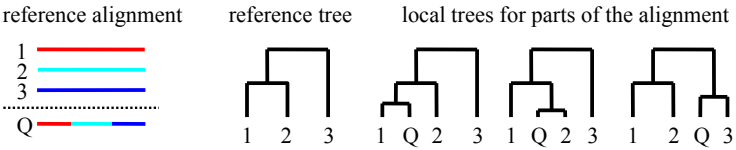


Figure 4.2: An illustration of the subtyping problem. To the left, the reference alignment with sequences 1, 2 and 3 is shown. The query sequence Q is displayed below the reference alignment and homology is indicated by different colors. The subtyping task is to determine the composition of the query sequence Q given a reference alignment and the corresponding reference tree. As subtypes are usually derived from current sequences in the reference alignment, the query sequence only involves recent recombination. In other words, the query sequence is derived from leaves in the reference tree, but not from older branches.

statistical test for significance. As there are so many possible breakpoints in an alignment, it is necessary to filter the relevant breakpoints first, or to correct for multiple testing.

3. Which sequences are recombinants?

If the recombinant sequences are removed, the resulting dataset evolved without recombination and can be analyzed for example by phylogenetic inference. But a more detailed analysis of the recombinant sequences is often also of interest. For example, the recombinants may show unusual properties in the lab, such as multi-drug resistance. In any case, the recombinant is often subject to further analysis as presented in question 4. Ideally, a method provides a *p*-value for recombination for each sequence in the alignment. As for question 2 it is necessary in principle to correct for multiple testing. But the number of sequences is usually well below 50 and much smaller than the number of possible breakpoints, so that the multiple testing problem is not a major concern. An inherent problem of finding the recombinant sequence is that the parental sequences also exhibit some recombinant structure (see Figure 4.1). It is difficult to infer which sequence is the true recombinant and which are sequences parental to the recombinant.

4. What is the structure or composition of a recombinant sequence?

Once a sequence has been identified as a recombinant sequence, the next step is to determine the parental sequences and the location of the recombination breakpoints. The information on the parental sequences often provides valuable insight into the properties of the recombinant. For HIV-1 the structure of the recombinant also suggests whether it is a novel recombinant form or a circulating recombinant form. Subtyping in HIV-1 is a special case of this problem, as the input sequence is usually derived directly from the sequences in the alignment, but not from the ancestors of the sequences (see Figure 4.2). Still, subtyping in HIV-1 is one of the most important questions in recombination analy-

sis today. Given that the input sequence is a known recombinant, a method should propose one or more hypotheses for the structure of the recombinant, possibly ordered regarding their likelihood. A measure of confidence for each hypothesis, such as a *p*-value, is also valuable.

5. What is the ARG underlying the sequences?

Inferring the ARG is the ultimate challenge in the analysis of recombinant sequences and gives complete information about recombination that took place in the dataset. The ARG answers all questions introduced before. Unfortunately, inferring the ARG is often impossible and intractable for all but the smallest datasets. Inferring ARGs is a more general case of phylogenetic tree inference, which is already an NP-complete problem by itself. The combinatorial space associated with inferring ARGs is much larger than that of phylogenetic tree inference, particularly if there are many recombination events. The large combinatorial space is also the reason why it is often impossible to infer a relevant ARG. The information in the sequence data often does not suffice to reconstruct the ARG unambiguously, particularly if the recombination rate is high. In this case, each local tree in the ARG only corresponds to a small region of the alignment and it is not possible to reconstruct the tree accurately. A high recombination rate requires a different approach for the analysis of recombination is not discussed here.

4.2 Common Strategies for Recombination Analysis

Methods for recombination analysis often implement similar strategies or make similar assumptions. It is often possible to describe these strategies and assumptions and discuss their consequences without referring to a particular implementation. First, I introduce several common assumptions regarding the input data and then discuss several strategies that are either common or of particular interest for recombination analysis.

A particularly interesting, but complex and usually intractable strategy is to infer the best ARG by maximum likelihood or maximum parsimony. Simpler strategies are based on the fact that ARGs define a set of local trees and that the evolutionary signal differs for the regions left and right of a recombination breakpoint. It is quite simple to scan for a change of the evolutionary signal by sliding a window over the sequence alignment. Several approaches detect recombination if the alignment is better described by more than a single tree, for example because trees reconstructed from adjacent parts of the alignment differ significantly.

4.2.1 Input Alignment

Many methods assume that the sequence dataset is given as a multiple alignment, but this is rarely a limiting assumption. As described in section 2.5 it is usually possible to construct a multiple alignment even if the sequences evolved subject to recombination. If a method does not require a multiple alignment as input, it usually only compares two se-

quences or implicitly constructs a multiple alignment. Interestingly, constructing the multiple alignment implicitly seems to increase the hypothesis space and results in a high computational cost (see e.g. the jpHMM method) or even a reduced power of detecting recombination.

4.2.2 Using Polymorphic and/or Synonymous Sites in the Alignment

Some methods, such as the Maximum Chi-Squared method (see section 4.3.1.1), detect recombination if mutations are not evenly distributed along the alignment. Without preprocessing, these methods detect recombination in any conserved region or mutation hotspot in the alignment. A simple, but effective strategy is to eliminate all constant columns from the alignment and only use polymorphic sites in the alignment as an input. The Maximum Chi-Squared method, for example, detects recombination only if a segment of polymorphic sites indicates that two sequences are more similar than expected. Sliding window-based methods often only use polymorphic or informative sites as input, too, to ensure that each window has about the same information content regarding the ancestry. It is important to realize that the analysis depends on the sequence context if only polymorphic or informative sites are used. Hence, the sequences in the alignment should be chosen with care. Finally, a simple and common strategy to counter the effect of selection is to restrict the analysis to synonymous sites in coding sequences as they are less affected by selection.

4.2.3 Assuming the Infinite Sites Model

The infinite sites model is a common assumption in population genetics and is also a prerequisite for methods that bound the number of recombination events (see section 4.3.8.2). The infinite sites model assumes that each mutation occurs at a different site in the genome. As there are no recurrent mutations in this case, each local tree fits to the mutation pattern perfectly and the total number of mutations in the ancestry of the sequences is equal to the number of polymorphic sites. As a result, the recombination parsimony problem (see section 4.2.7) does not have to penalize mutations, but only has to minimize the number of recombinations in the inferred history. Even though inferring the ancestry is easier in this case, it is still very difficult and a variant of the problem where ancestral sequences are given has been claimed to be NP-hard (Song, Wu et al. 2005).
Unfortunately, the infinite sites assumption is often violated for real sequence data, unless there are few polymorphisms in the alignment. If recurrent mutations exist, methods based on the infinite sites assumption usually infer artificial recombinations such that the model fits the data. High mutation rates or mutation hotspots can therefore lead to falsely detected recombination events.

4.2.4 The Column Permutation Test

Many methods only compute an arbitrarily scaled indicator for recombination, but do not provide *p*-values. The absolute value of these indicators usually depends on basic proper-

ties of the alignment, such as number of sequences, sequence length or sequence divergence and is not indicative for recombination. Fortunately, it is often possible to transform an indicator for recombination into a *p*-value using a simple permutation test. To be more specific, the null hypothesis for the case of recombination detection is that the alignment evolved without recombination. A permutation *p*-value can be computed if it is possible to sample alignments from the null hypothesis, while retaining the basic properties of the alignment. The *p*-value for recombination is the probability that the original indicator value was observed, given that indicator values are distributed as for alignments of the null hypothesis.

A common approach is to simulate alignments by permuting the columns of the alignment, as described for example in (Wiuf, Christensen et al. 2001). Permuting the columns of an alignment does not change the basic properties of the alignment, such as sequence divergence, but eliminates linkage between sites. In other words, the sequence of local coalescent trees is also permuted for the permuted alignment. For an alignment that evolved without recombination, there is only one coalescent tree for all sites, and a column permutation does not change the tree. But if there is one recombinant in the alignment, there are two local coalescent trees, one for the left part and another for the right part of the alignment. The column permutation for such an alignment distributes the local coalescent trees across the alignment and effectively decorrelates the local coalescent trees. A possible ancestry for the permuted alignment therefore involves a high number of recombination events. Finally, if the alignment evolved with an infinite amount of recombination, all sites in the original alignment are unlinked. A column permutation does not change the expected distribution of alignments in this case. Overall, the null hypothesis corresponding to column permutations is that either all sites are linked and there is no recombination, or all sites are unlinked and evolved according to an independent history. The column permutation test only works for methods for analyzing recombination that respect the ordering of sites during the analysis. For example, if a method only uses the global distances between sequences as input, the input does not change by a column permutation and the permutation test cannot be used.

It might seem counterintuitive to test for recombination by comparing an alignment with a few recombinations to alignments that were created with a high number of recombinations. But there is a simple justification. If there is a high number of recombination in the alignment, all sites are unlinked and the information content on recombination is much lower than for the case of a single recombination. A method would therefore rather assume that the local coalescent trees vary due to noise and not because an enormous amount of recombination could be inferred. Consequently, the indicator for recombination usually suggests little recombination in this case.

The column permutation test is very simple and general enough to apply for many different methods. As most methods rely on the column permutation test to compute *p*-values, exceptions from this rule are explicitly stated in the description of the method. There are

two related shortcomings for the column permutation test, though. First, it is necessary to compute the indicator for every permuted alignment. The permutation test therefore requires considerable computing time and cannot realistically be used for compute intense methods. Second, the smallest possible *p*-value depends on the number of permutations, but the number of permutations increases the computing time linearly. If there are 10^n permutations, the smallest *p*-value is roughly $1/10^n$. If *p*-values below 0.05 are considered significant, it is usually sufficient to compute *p*-values based on 100 to 200 permutations, and the permutation test is still feasible. The number of permutations can increase dramatically if the *p*-values have to be corrected for multiple testing, for example because several sequences or several putative breakpoints were tested for recombination. A common approach is to use Bonferroni correction and multiply the *p*-values with the number of tests performed before their significance is evaluated. In other words, if *p*-values are corrected for ten tests, only *p*-values below 0.005 are significant before correction and it is already necessary to perform 1000 to 2000 permutations to achieve an appropriate accuracy.

4.2.5 Based on a Sliding Window

A recombination results in different evolutionary trees left and right of the recombination breakpoint. Hence, many methods use a sliding window and compute some measure depending on the tree for the left and the right part of the sliding window. If the two measures suggest that the trees are different, a recombination breakpoint is inferred at the center of the sliding window. Many methods condition the test on a query sequence in the alignment, such that recombination breakpoints are only detected in the query sequence. As a simple example, recombination in the query sequence can be detected if the closest sequence to the query sequence is different in the left and right part of the sliding window. Methods based on a sliding window usually can compute *p*-values for recombination breakpoints using a modified version of the column permutation test. In contrast to the original column permutation test, only the columns inside the sliding window are permuted. However, all approaches based on sliding windows share a major problem. It is often difficult to set the size of the sliding window appropriately. If the sliding window is too small, the measure depending on the tree is not stable and spurious recombination events may be inferred. If the sliding window is too large, it may cover more than one recombination breakpoint. In this case, there is no correct tree for the left or right part of the sliding window and the method may fail to detect the recombination event as the estimated trees are arbitrary. A large window size also leads to a poor spatial resolution of the recombination breakpoint, as neighboring windows share much of the signal if they are overlapping. Hence, several recombination breakpoints in close proximity are detected if there is no additional post-processing step. Finally, as adjacent windows are usually overlapping, each part of the alignment is evaluated more than once and a sliding window based approach adds quite some computational cost.

4.2.6 Based on Comparisons of (Unrooted) Tree Topologies

Several methods reconstruct trees for different parts of the alignment and infer recombination if the reconstructed trees differ. There are several implications of such an approach. First, tree topologies in adjacent regions are similar. Reconstructing the trees in adjacent regions independently introduces more parameters than theoretically necessary and the trees may be less stable. Second, only informative sites provide information on the tree topology if the method uses parsimony, compatibility or another simple scoring scheme to reconstruct trees. Non-informative sites are ignored by these methods even though they may contribute information on past recombination event. Third, there is only one unrooted topology for three sequences, and therefore there must be at least four sequences in the alignment to detect an unrooted topology change. Finally, most methods based on comparisons of unrooted tree topologies have to define when a topology change is significant. As the tree is inferred from a limited region in the alignment, the inferred tree may be unstable and topology changes can occur by chance. A common approach is to use a measure of robustness, for example the bootstrap value (Bootscanning in section 4.3.4.1) or the posterior probability of a topology, and detect topology changes depending on the robustness of the inferred trees. An inherent problem remains even if the local trees are correctly inferred: a post-processing algorithm has to extract the information on recombination events from the trees, for example by minimizing the number of SPR operations between the trees.

4.2.7 The Recombination Parsimony Problem

A widely used approach to reconstruct phylogenetic trees is that of maximum parsimony. Given a sequence alignment, maximum parsimony finds the tree topology connecting the sequences such that the number of mutations along the branches is minimized. A similar criterion can be used to reconstruct the history of a sequence alignment subject to mutations and recombinations (Hein 1990; Song and Hein 2005). The cost function then depends on the recombination cost r, the local topology τ_p for each position p, the mutation cost $m(p,\tau_p)$ induced by the topology τ_p at position p and a distance measure d for topologies. The corresponding optimization problem is called recombination parsimony and solves

$$\arg\min_{\tau_1,\ldots,\tau_N}\left\{r\sum_{p=1}^{N-1}d\left(\tau_p,\tau_{p+1}\right)+\sum_{p=1}^{N}m\left(p,\tau_p\right)\right\} \tag{4.1}$$

The recombination parsimony problem requires that the distance d counts the number of recombinations necessary to transform two trees into each other. As described in section 3.3.3.2, the number of recombinations is equal to the number of SPR operations on rooted, ordered binary trees (Song and Hein 2003; Song 2006). The topologies τ_p therefore have to be rooted, ordered binary trees for the recombination parsimony problem. Obviously, the state space of recombination parsimony is huge, as the topology may vary for every position and the number of topologies grows super-exponentially with the

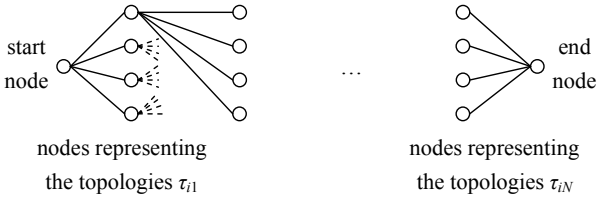


Figure 4.3: The dynamic program that solves the maximum parsimony problem with recombination is shown, but some transitions are left out for clarity. Each column corresponds to a column in the alignment and all nodes in a row correspond to the same topology. Therefore, a transition between nodes in different rows represents a recombination event and is weighted according to the SPR distance between the topologies. The node in row i and column p is weighted by the mutation cost $m(p,\tau_{ip})$ and depends on the topology τ_{ip} that the node represents. The cost optimal path connecting the start and the end node describes a solution for the recombination parsimony problem.

number of sequences (Felsenstein 1978). The recombination parsimony problem is a more general version of the NP-complete Steiner problem in phylogeny (Foulds and Graham 1982) and can only be solved exactly for very few sequences and positions.

Hein (Hein 1990; Hein 1993) describes a method for recombination parsimony that performs far better than a brute force approach, but of course cannot mitigate the inherent complexity of the problem. The important observation is that d only depends on pairs of adjacent topologies τ_p and τ_{p+1}, but does not model any long range effect. Consequently, recombination parsimony can be solved by dynamic programming. The tableau f_{pt} of the dynamic program stores the best solution up to position p depending on the topology t for position p (see Figure 4.3). The recursion equations are:

$$\begin{aligned} f_{1,t} &= 0 \\ f_{p,t} &= \min_{t' \in \mathbf{T}} \left\{ f_{p-1,t'} + m(p,t) + d(t,t') \right\} \end{aligned} \tag{4.2}$$

where $\mathbf{T}$ is the set of all possible tree topologies. The dynamic program requires $O(NT^2)$ operations, where T is the number of rooted, ordered binary topologies, and is not practical for more than a few sequences.

Given the solution of the recombination parsimony problem, it is easy to extract the answer for all questions of recombination analysis. As recombination parsimony is computationally intractable for all but very small sequence alignments, it is still mainly interesting in a theoretical context and as a basis for developing heuristic algorithms.

4.3 Methods for Recombination Analysis

The following section describes several methods for the analysis of recombination. A comparison regarding output, recombination detection performance, inherent limitations of the method and speed is provided later (see section 4.4). The output may solve any of

the five questions introduced before in section 4.1 and should ideally also provide a measure of confidence, such as a *p*-value. It is also important to compare the accuracy of the output for the different methods. Unfortunately, the validation in the original publications is often useless for this purpose, as methods are usually validated on different alignments and under different conditions. There are two publications that compare the power of detecting recombination for several methods (Posada and Crandall 2001; Wiuf, Christensen et al. 2001) and these are used as a reference for comparing the accuracy of the methods in the following. As a result, the comparison does not cover all methods and only refers to the task of recombination detection. More details on the setup for a comparable power study are given in section 7.1. Particularly interesting for practical reasons is also the running time of the method, as it may vary from seconds to days for a typical alignment with 10–20 sequences and 1,000–10,000 sites.

There are numerous methods for recombination analysis and a complete description of all these methods is out of scope for this thesis. The focus of this section is on methods comparable to the method developed later in this thesis. Most of these methods are fast and widely used or at least of potential interest for a biologist or virologist. Many other methods are also briefly introduced to cover a broad range of strategies, but they are not discussed as comprehensively. For example, coalescent methods are extremely slow and difficult to set up. Several other advanced methods even define a restricted version of recombination parsimony and solve it optimally. Unfortunately, most of these methods do not provide a measure of robustness for their output and are not thoroughly validated.

For sake of clarity, all methods are presented using the same notation as far as possible. In the following, N is the number of sites in the alignment, M is the number of sequences, p=1,…,N denotes positions in the alignment and w is the window size if a sliding window is used. Furthermore, the multiple alignment is represented as a matrix $\mathbf{A}$ with elements A_{ip} and columns A_p.

4.3.1 Methods Testing the Distribution of Mutations

The following methods are based on statistical tests that check for an unequal distribution of mutations in the alignment. The *p*-value is then computed with the column permutation test. As all methods are based on simple test statistics, they are extremely fast. Nevertheless, the power study by Posada and Crandall (Posada and Crandall 2001) showed that these methods are among the best methods for detecting recombination in an alignment. A more detailed analysis of recombination using these methods is not always possible as they may not compute a *p*-value for recombination in a sequence or for a recombination breakpoint. It is also difficult to interpret their output in terms of estimating the structure of a recombinant sequence.

4.3.1.1 Maximum Chi-Squared (MC^2) Method

Initially the Maximum Chi-Squared (MC^2) method (Smith 1992) was described as a hand-driven protocol that analyzes a single sequence for a recombination signal, but it can also be used in an automated procedure. The input is usually a multiple alignment in which all sites that are not polymorphic are removed. The MC^2 method detects recombination if the differences between two sequences are not uniformly distributed along the alignment, but cluster in one part of the alignment. The MC^2 method is based on the 2x2 chi-square test statistic c (Spencer 2003): suppose there are n polymorphic sites in the alignment and position k is tested for a recombination signal. If the two sequences differ at s sites and there are r differences to the right of position k, the chi-square statistic is computed as:

$$c = \frac{n(ks - nr)^2}{ks(n-k)(n-s)} \tag{4.3}$$

In essence, the chi-square test compares the observed number of differences to the right of position k with the number of differences that are expected if they were distributed equally along the sequence. The chi-square statistic c attains a high value if the differences significantly cluster to the left or to the right of position k. The most significant breakpoint in the alignment is the position k that maximizes the value of c. The MC^2 method then computes a p-value for the maximum value of c by the column permutation test. An exact method for computing the p-value is also available (Spencer 2003). The p-value is the probability of observing such an uneven distribution of differences by chance and therefore represents the p-value for recombination in the evolution of the sequence pair. It is straightforward to extend the MC^2 for recombination detection to the whole alignment: the maximum value of c is computed over all sequence pairs and all positions k. A similar approach could be used to compute a p-value for the recombination breakpoint. A fundamental problem of the MC^2 method is that it only checks for a significant bipartition of the alignment. If a sequence evolved by more than one recombination or by gene conversion, the MC^2 method is suboptimal and can fail. Maynard Smith proposed a manual work-around to this problem (Smith 1992).

4.3.1.2 Geneconv

Geneconv (Sawyer 1989) (http://www.math.wustl.edu/~sawyer/geneconv/) uses polymorphic sites in the multiple alignment as an input and detects recombination if two sequences are more similar in a region than expected. For each region a score is computed to reflect similarity: matches count as +1 and mismatches count as $–x$ where x is a user-specified parameter. Geneconv detects recombination in a sequence pair if the maximum score over all possible regions in the sequence pair is significantly large. The p-value for recombination in the sequence pair is either obtained with a permutation test or by analytical means. Geneconv can also calculate p-values for recombination in the whole alignment by computing the maximum score over all sequence pairs and all regions. The

p-value is again obtained by the column permutation test. As Geneconv only provides a list of significantly scoring regions, it is difficult to infer the structure of a recombinant sequence. In contrast to the MC^2 method, Geneconv does not depend on a bipartition of the alignment and works well even if a sequence contains more than one recombination.

4.3.1.3 PhylPro

PhylPro (Weiller 1998) computes the distance vector of the query sequence to all other sequences for the left and right part of each sliding window. The correlation between the left and right part of the sliding window is computed and plotted along the alignment. A low correlation between the distances in the left and right part of the window indicates a recombination breakpoint, as the evolutionary trees for the left and right part of the window then are usually different. Usually, every sequence in the alignment is taken as the query sequence in turn. Thus, there are *N* different lines in the plot in total, one for each possible query sequence. The lines in the plot showing a strong recombination signal correspond to the recombinant sequences. The column permutation test could be used for computing *p*-values, but PhylPro does not implement it.

PhylPro is a fast and simple algorithm and is rated as one of the better methods for detecting recombination if *p*-values are computed by the column permutation test (Posada and Crandall 2001). A major problem of PhylPro is that it takes quite some experience to interpret the plots correctly, as the result strongly depends on the sequences in the alignment. The author suggests selecting the set of sequences carefully by hand to obtain better signals for a detailed analysis of recombination.

4.3.2 Compatibility Methods

Two sites are called compatible if a tree describing their evolution does not contain any recurrent mutation. If sequences only consist of 0's and 1's, the criterion is very simple: two sites are compatible if there only occur three of the four possible states {00, 01, 10, 11} in every sequence at the two sites (see Figure 4.4). A similar criterion can be defined if each site has more than two states. Pairwise compatibility is usually represented as a matrix **C** where an element C_{ij} is 1 if site *i* and *j* are compatible with each other and 0 otherwise. As only informative sites can lead to incompatibility between sites, methods that are based on compatibility usually remove all non-informative columns from the alignment. A site *p* is informative if the column A_p contains at least two different states and each state occurs at least twice. In this case, the informative site partitions the sequences into clusters of two or more sequences and contains information about the evolutionary tree.

Compatibility can be used to detect recombination, as a recombination event usually leads to incompatible sites to the left and right of the recombination breakpoint. To be more specific, if there are no recurrent mutations and only informative sites are used as input, sites to the left of the recombination breakpoint are incompatible with sites to the

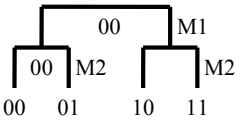


Figure 4.4: The tree minimizes the number of mutation events and shows all ancestral sequences and all mutation events that are required to generate the four sequences at the leaves of the tree. Only two states are allowed at each sequence position. Mutation events start with M and are followed by the position that is changed. As the two sites are incompatible, the tree infers a recurrent mutation event M2.

right of the recombination breakpoint, but all pairs of sites to the left or right are compatible with each other. However, incompatibilities can also arise if the infinite sites model is violated and recurrent mutations occur.

Methods based on compatibility are usually very fast as they compute the compatibility matrix once and then derive simple statistics based on it. Even though they are fast and easy to implement, compatibility methods belong to the best methods for detecting recombination in an alignment regarding accuracy (Posada and Crandall 2001; Bruen, Philippe et al. 2006). Nevertheless, compatibility is inherently limited regarding the analysis of recombination. Compatibility can only detect recombinations that change the unrooted topology of the local coalescent trees. A more serious limitation is that compatibility methods cannot detect the recombinant sequences in the alignment. Also, no method presented here detects recombination breakpoints even though, in principle, breakpoints could be inferred from the compatibility matrix. Compatibility methods are therefore best used as a fast scan for detecting recombination, before a more detailed analysis is conducted with other methods.

4.3.2.1 Neighbor Similarity Score (NSS)

The NSS (Jakobsen and Easteal 1996; Jakobsen, Wilson et al. 1997), also known as Reticulate, computes the fraction of elements in the compatibility matrix with the same value as the neighboring element

$$NSS = 1 - \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1}^{N-1} \left| C_{ij} - C_{i,j+1} \right| \tag{4.4}$$

where N is the total number of sites in the alignment. The NSS measures whether the compatibility matrix shows some clustering of 0's and 1's. The NSS is closer to 1, if 0's and 1's occur in blocks in the matrix. Two adjacent blocks are indicative of a recombination event, as a set of adjacent sites is incompatible with a set of other adjacent sites. The p-value for recombination in the alignment is computed by the column permutation test.

4.3.2.2 Pairwise Homoplasy Index (PHI)

The PHI test (Bruen, Philippe et al. 2006) defines a quantitative version of incompatibility: the incompatibility score of two sites is the minimum number of recurrent mutations needed to explain the two sites by a tree. In the following, C_{ij} refers to the incompatibility matrix that tabulates the incompatibility score for every pair of informative sites i and j. The PHI test computes the average of the incompatibility score, sliding a window of size w (measured in informative sites) across the alignment. To be more specific, the test statistic is defined as

$$\Phi = \frac{2}{w(2n-w-1)} \sum_{i=1}^{n-w-1} \sum_{j=i+1}^{i+w+1} C_{ij} \tag{4.5}$$

where n is the total number of informative sites in the alignment. A small value of Φ indicates that all sites within a window are usually compatible. The distribution of Φ can be approximated, but it is also possible to use the column permutation test for computing the p-value for recombination in the alignment. A significant p-value suggests that adjacent sites on average have a significantly higher incompatibility score than a random set of sites. The PHI test is more robust w.r.t. mutation rate heterogeneity and exponential population growth than the NSS test or the MC^2 test. The window size introduces an additional parameter, and the p-value for recombination depends on it.

4.3.3 Methods for Subtyping

Subtyping is an important problem in HIV-1 sequence analysis. The goal of subtyping is to determine the structure of a possibly recombinant query sequence with respect to a reference alignment of non-recombining sequences. In other words, the task is to find the corresponding subtype in the reference alignment for every position of the query sequence. The separation between subtyping methods and other methods for recombination analysis is often quite artificial. A subtyping method can usually also process an arbitrary alignment and take each sequence in the alignment as a query sequence in turn to analyze the full alignment. The result of such an analysis identifies recombination in the alignment, recombinant sequences and also predicts recombination breakpoints. However, some subtyping methods also require as input a phylogenetic tree for the sequences in the reference alignment and are therefore limited to sequence alignments that only contain a single recombinant sequence. Furthermore, subtyping methods often are only available via a website and take the query sequence as input, but assume a fixed reference alignment. As a result, the implementations of subtyping methods are often not suitable for a more general analysis of recombination. A more general analysis requires that the program allows the user to specify the reference alignment and that the program tolerates recombinant sequences in the reference alignment. Finally, most subtyping methods implement very simple strategies and do not work well for recombination analysis in general.

4.3.3.1 RIP

RIP (Siepel, Halpern et al. 1995) (http://www.hiv.lanl.gov/content/sequence/RIP/RIP.html) is the subtyping program of the Los Alamos National Laboratory website. As a first step, RIP computes the distance d_{ip} between the query sequence and sequence i in the reference alignment for every sliding window position p. Then, RIP compares the distance of the best matching reference sequence i to the second best matching reference sequence i' and computes the z-score of the difference $d_{i'p}$-d_{ip} for every window position p. If the z-score is significant, the best matching reference sequence is the predicted subtype for window position p. The query sequence is a recombinant if there are two or more predicted subtypes for different window positions. RIP also visualizes the distances d_{ip} by plotting them along the alignment.

RIP is a simple and fast approach to scan a query sequence for recombination. RIP can even predict the structure of the putative recombinant sequence and provides a simple measure of confidence. It is probably safe to assume that RIP performs worse than Bootscanning or other more complex sliding window approaches, even though a thorough evaluation has never been done.

4.3.3.2 STAR

STAR (Gale, Myers et al. 2004; Myers, Gale et al. 2005) is a subtype analysis program specifically designed for the pol region of HIV-1. STAR derives positional amino acid frequency profiles for each subtype and for the whole alignment using the subtype reference alignment of the Los Alamos HIV Database (http://www.hiv.lanl.gov/). For each position p of the query sequence and each possible subtype i, an odds ratio o_{pi} is computed as the probability that the query is derived from the subtype profile as compared to the alignment profile. An overall score S_i is computed using two thresholds a and b

$$S_i = \sum_{p=1}^{N} o_{pi} > a \Big/ \sum_{p=1}^{N} o_{pi} < b \tag{4.6}$$

where $o_{pi}>a$ is 1 if o_{pi} is greater than a and 0 otherwise. The predicted subtype of the query is the subtype i for which the score S_i is maximal. A z-score is computed from the S_i by assuming that the S_i are normally distributed. The thresholds a and b are optimized on the subtype reference alignment, such that they result in a high z-score for correctly classified sequences. Recombination is detected if the query sequence results in a z-score below 2.5. In this case, the fit of the query sequence with each subtype profile is computed using a sliding window and compared to the fit with the predicted subtype. Obviously, STAR is a very simple method and cannot be expected to perform better than Simplot (section 4.3.4.1) or other sliding window methods.

4.3.3.3 Jumping Alignment

Jumping alignments (Spang, Rehmsmeier et al. 2002) is a method for remote homology detection in sequence database searches. Even though jumping alignments were never

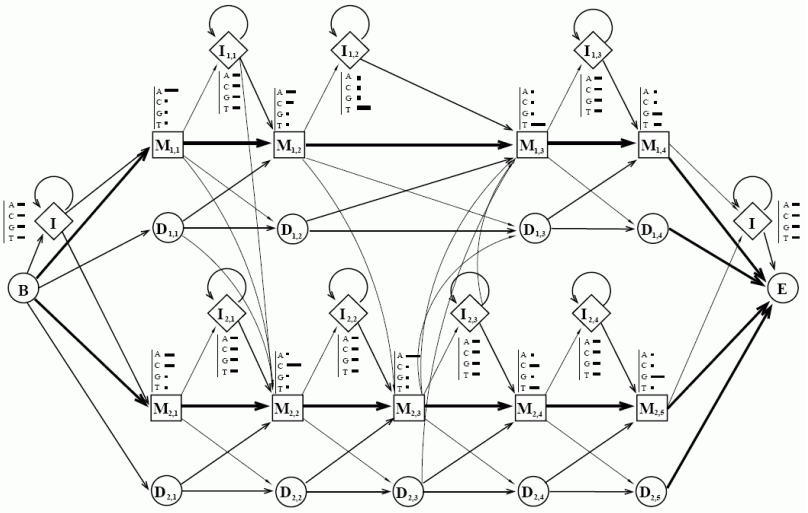


Figure 4.5: The structure of the jpHMM model for two short subtype sequences. Not all transitions are shown. The upper and the lower part of the HMM correspond to a profile HMM for each subtype, where M, D and I are match, deletion and insertion states, respectively. A transition between the two profile HMMs may never jump backwards such that a tandem-repeat would be created. The Figure is taken from (Schultz, Zhang et al. 2006) and is used under the Creative Commons Attribution Licence.

used for subtyping, they introduce an optimization problem that is closely related to the subtyping problem and also inspired the jpHMM approach for subtyping (see below). A jumping alignment is essentially a pairwise alignment of a query sequence with the columns of a reference alignment and therefore matches each position of the query sequence with a column in the reference alignment. The jumping alignment can introduce gaps in the query sequence or columns of gaps in the reference alignment, but cannot introduce gaps in a single sequence of the reference alignment. The mismatch cost for pairing a position of the query sequence with a column of the reference alignment is computed as follows. At each position only a single sequence of the reference alignment is used to determine the mismatch cost. A change of the selected sequence in the reference alignment between adjacent positions is penalized and corresponds to a recombination event. Overall, the nucleotides along the path of selected sequences in the reference alignment define a recombinant sequence that is aligned to the query sequence. Hence, a jumping alignment assigns a reference sequence to every position in the query sequence and can be interpreted as the subtype structure of the query sequence. A jumping alignment depends

on the cost for mismatches, indels and the cost for jumps between selected reference sequences. The cost-optimal jumping alignment is computed by dynamic programming in time proportional to $O(MN^2)$ if query sequence and reference alignment are of length N.
A jumping alignment not only infers the positions of the recombination breakpoints, but also provides detailed information on the (possibly recombinant) structure of the query sequence. For an application in the field of subtyping, there are two important additional requirements. First, the mismatch and indel cost, and the cost for recombination have to be set carefully such that the method performs well for subtyping. Second, the method only outputs the structure of the query, but does not provide any *p*-values for recombination, for breakpoints or for the structure. Some measure of robustness of the output would be extremely valuable.

4.3.3.4 Jumping profile Hidden Markov Model (jpHMM)

JpHMM (Schultz, Zhang et al. 2006; Zhang, Schultz et al. 2006) is an extension of jumping alignments and is designed especially for subtyping. A profile HMM models a distribution of sequences and is composed of match states, insertion and deletion states. For a given sequence, it is possible to determine the path through the states that maximizes the probability that the sequence was observed. JpHMM models each subtype with a separate profile HMM, but also allows for transitions between the profile HMMs for different subtypes, similar to the jumping alignment method (see Figure 4.5). Transitions between the profile HMMs are restricted to destination positions equal to or forward of the source position, such that the generated sequences do not contain a repeat. Emission probabilities and transition probabilities within a subtype can be estimated from the subtype reference alignments. The transition probability (or probability for recombination) between subtypes has to be set *a priori*, as the reference alignments do not contain any inter-subtype recombinant sequence and therefore do not provide any information on the probability of a transition between subtypes. Given a query sequence, the path with the highest probability determines the subtype structure of the query sequence.
JpHMM is not suitable as a general method for recombination analysis, as it requires a set of sequences for each subtype to estimate the parameters of the model correctly. JpHMM shares the drawbacks of the jumping alignment approach as it does not provide *p*-values or another measure for the robustness of the solution. The strength of jpHMM is clearly that it handles gaps easily and very accurately. On the other hand, the large number of transitions for insertion and deletion events leads to a time complexity quadratic in the sequence length N. As a result, the running time is slow compared to other approaches even if the path with the highest probability is only approximated with a heuristic.

4.3.4 Methods That Compare Local Trees and Use a Sliding Window

The most frequent strategy in recombination analysis is to compare local trees that are inferred in a sliding window across the alignment. In this case, it is not only difficult to

set the size of the window (see section 4.2.5), but the method also has to define when a change in local tree topology is significant. This category is very popular among researchers, probably because tree inference is well understood and accepted. But the accuracy of the resulting recombination analysis methods is quite poor. Two methods in this category, Bootscanning and PLATO, were evaluated in the power study (Posada and Crandall 2001) and performed worse than many other methods. Interestingly, the sliding window based approach PhylPro was among the best methods in this study. PhylPro is quite similar to Bootscanning, as it extracts distance vectors from the left and right part of the window, but then it directly compares these vectors without reconstructing a tree. This indicates that the tree reconstruction step may hurt more than it helps. In essence, there are two potential sources for errors in a tree-based approach: the tree reconstruction method itself and the method used to compare tree topologies. Both error sources probably play a role in the failure of Bootscanning in the power study. On the positive side, reconstructing trees usually allows for inferring the parental sequences and the structure of a recombinant query sequence.

There are many other methods that fall into this category, but are not discussed in detail below. VisRD (Strimmer, Forslund et al. 2003; Forslund, Huson et al. 2004) for example is based on quartet-mapping and computes the quartets for each sliding window. VisRD then detects recombination solely based on a visual inspection of two summary statistics that aggregate the topologies of the quartets. VisRD does not provide a single measure indicating recombination and is also not evaluated quantitatively on simulated data. RDP (Martin and Rybicki 2000; Martin, Williamson et al. 2005) detects recombination using a sliding window approach on sequence triplets if the two sequences that are most similar over the whole alignment are not as closely related to each other as the third sequence to any of the two for some region in the alignment.

4.3.4.1 Bootscanning (Simplot)

One of the most popular methods for the analysis of recombination is Bootscanning (Salminen, Carr et al. 1995), also known by the name of the widely used implementation Simplot (Lole, Bollinger et al. 1999). Bootscanning is based on phylogenetic tree inference and bootstrap support values. Bootstrap support values are a measure of confidence for the edges of an inferred phylogenetic tree. To be more specific, cutting an edge in a phylogenetic tree separates the set of leaves in the tree into two sets and can therefore be represented as a bipartition. The bootstrap support of a bipartition or edge in the tree is the percentage of bootstrapped alignments that result in a phylogenetic tree with the same bipartition. Each bootstrapped alignment is generated by randomly sampling the columns of the alignment with replacement. In essence, bootstrapping generates alignments that only contain on average 60% of the columns of the original alignment (Hastie, Tibshirani et al. 2001) and a high bootstrap support therefore indicates that the bipartition is robustly inferred.

Bootscanning usually only analyzes a single query sequence in the alignment and seeks to infer the structure of the query sequence. As a precomputation step, bootscanning infers a phylogenetic tree with bootstrap support values for each sliding window. The phylogenetic trees are often inferred by a distance method, but maximum likelihood is also possible. Then, bootstrap support values for the bipartition pairing the query sequence with every other sequence in the alignment are plotted for every sliding window position. The result is a plot that shows for each sliding window a measure of similarity of the query to any other sequence in the alignment. Significant evidence for recombination is detected if there are at least two windows in which the query pairs with a different sequence and the bootstrap support values are higher than 70%. Recombination breakpoints are inferred at the positions where the lines indicating bootstrap support cross in the plot. The column permutation test can be used to compute *p*-values for recombination in the alignment, in a sequence or at a putative breakpoint, but this is usually not part of the analysis and is not implemented in Simplot. Simplot also implements another method for visually scanning for recombination. Sliding a window over the alignment, the similarity of the query with every other sequence in the alignment is plotted. As before, *p*-values are not computed by Simplot.

The output of Bootscanning and of the similarity method of Simplot is difficult to interpret. The signal is usually very noisy, particularly if there are many sequences in the alignment. The authors suggest including only the potential parental strains in the analysis to eliminate noise, but this requires manual interaction and an interpretation of the noisy signal. Despite its popularity, Bootscanning was one of the worst methods in the power study (Posada and Crandall 2001).

4.3.4.2 SlidingBayes

SlidingBayes (Paraskevis, Deforche et al. 2005) uses the same basic idea as Simplot to analyze sequences regarding recombination. The main difference to Simplot is that SlidingBayes infers the phylogenetic trees for each sliding window with a Bayesian method and not with a distance or maximum likelihood method. Instead of the bootstrap support, SlidingBayes plots the posterior probability of a partition of the sequences. Overall, SlidingBayes does not differ significantly from Simplot and inherits all advantages and disadvantages. As phylogenetic trees in Simplot are usually estimated with a distance method, SlidingBayes is usually more accurate, but also a lot slower.

4.3.4.3 Partial Likelihoods Assessed Through Optimization (PLATO)

PLATO (Grassly and Holmes 1997) is a method for scanning a sequence alignment for different phylogentic evidence in separate regions of the alignment, so-called spatial phylogenetic variation. PLATO was also proposed as a method for detecting recombination by the authors. PLATO first finds the maximum likelihood phylogeny for the whole alignment. Given the maximum likelihood phylogeny, it is possible to compute the like-

lihood L_i for each site i, as phylogenetic inference usually assumes that each site evolves independently. The idea is then to find regions that are characterized by a low average likelihood. The statistic computed for every window position p and several window sizes w is

$$Q_{wp} = \frac{\sum_{i=p}^{p+w-1} \ln L_i \Big/ w}{\left(\sum_{i=1}^{p-1} \ln L_i + \sum_{i=p+w}^{N} \ln L_i\right) \Big/ N - w} \tag{4.7}$$

The numerator and denominator compute the average log-likelihood inside and outside the window, respectively. A large negative numerator represents a low likelihood, as log-likelihoods are always negative. Large values of Q_{wp} therefore indicate that the sites at positions p to p+w-1 do not fit to the global tree very well. Recombination is inferred if the maximum value of Q_{wp} over w and p is significantly large. PLATO introduces two methods to obtain p-values for the maximum value of Q_{wp} and detect recombination in the alignment. As the maximum value of Q_{wp} is approximately normally distributed, an analytical p-value for recombination in the alignment can be computed. Alternatively, it is possible to compute p-values based on a permutation test: as the global phylogeny has been estimated, alignments without recombination are generated by simulating sequences along the global phylogeny. The same procedure as above is then used to compute a p-value for the maximum of Q_{wp}.

PLATO assumes that there is an evolutionary tree for the majority of sites in the alignment and that this evolutionary tree can be recovered. Consequently, PLATO is inherently limited to small recombinant regions. As the recombinant regions grow larger, there is no predominant tree anymore and PLATO can fail to detect recombination. PLATO also does not perform well compared to other methods in the power study (Posada and Crandall 2001).

4.3.4.4 TOPAL

TOPAL (McGuire and Wright 1998; McGuire and Wright 2000) is similar to PhylPro, as it compares the evolutionary signal in the left and right part of the sliding window. But in contrast to PhylPro, TOPAL includes a tree estimation step and does not depend on specifying a query sequence. TOPAL first computes the distances l_{ij} and r_{ij} between sequences i and j in the left and right part of the window and then normalizes the distances to the same mean. TOPAL then reconstructs an evolutionary tree from l_{ij} and recovers the distances t_{ij} encoded in the evolutionary tree. The DSS statistic is computed as

$$DSS = \left| \sum_{i,j} (l_{ij} - t_{ij})^2 + \sum_{i,j} (r_{ij} - t_{ij})^2 \right| . \tag{4.8}$$

Small values of DSS indicate that the tree is a good model of evolution for the left and right part of the sliding window. If there is a recombination breakpoint in the middle of the sliding window, the tree reconstructed from the left part encodes different distances than that of the right part and DSS is large. Using the column permutation test, TOPAL

can compute p-values for recombination in the alignment and for the recombination breakpoint. The recombinant sequence cannot be recovered, though. The recombination detection performance of TOPAL was not compared to other methods in a power study, but as it is a mixture between PhylPro and Bootscanning it should perform similar to one of the two methods.

4.3.4.5 Probabilistic Divergence Method (PDM)

The basic idea of PDM (Husmeier and Wright 2001) is to compute the distribution of phylogenetic tree topologies for each window position in the alignment and detect recombination if the topology changes. To be more specific, PDM approximates the posterior probability $P_{\tau p}$ for every window position p and every possible topology τ by Markov chain Monte Carlo. The posterior probability Q_τ of the global topology is obtained by averaging $P_{\tau p}$ over all p. Similar to PLATO, a recombination is detected globally if the distribution $P_{\tau p}$ of topologies at some position p is significantly different from Q_τ. The distributions $P_{\tau p}$ and Q_τ are compared using the Kullback-Leibler divergence and p-values can be derived analytically. PDM also uses a local measure for detecting recombination: PDM detects recombination at position p if $P_{\tau p}$ and $P_{\tau,p+1}$ are significantly different. PDM is severely limited by computation time and the number of sequences in the alignment, as the number of topologies τ grows super-exponentially with the number of sequences (Felsenstein 1978). The recombination detection performance of PDM is only slightly better than TOPAL and usually does not justify the high computational cost.
A recent extension (Kedzierska and Husmeier 2006) improves the PDM by modeling the transitions between topologies more accurately. A small change in topology is often not significant, for example changes involving closely related sequences. Hence, the new method only records the posterior probability of a cluster of topologies. However, the most important improvement is that a hidden Markov model is used to describe the sequence of topology clusters in a similar fashion as the HMM approach (see section 4.3.5.3).

4.3.5 Methods that Compare Local Trees and Optimize Breakpoints

The methods introduced in the previous section use a sliding window to cut the alignment into regions that are then compared concerning changes of local topology. However, an inherent problem of sliding window-based approaches is the window size, as it limits the accuracy of the estimate. It is desirable to eliminate the sliding window and to use a more accurate partitioning of the alignment into non-overlapping segments. Unfortunately, the number of possible partitions N^K of the alignment grows exponentially with the number of putative breakpoints K. Hence, only few methods try to optimize the partitioning of the alignment and compare trees on these partitions. These methods often only output a sequence of unrelated trees. It is difficult to determine the cause for the changes and identify the recombinant sequence or its structure.

4.3.5.1 Genetic Algorithm for Recombination Detection (GARD)

GARD (Pond, Posada et al. 2006; Pond, Posada et al. 2006) is based on a model selection technique and compares the hypothesis of no recombination with the hypothesis of at least one recombination. First, GARD estimates a phylogenetic tree topology by neighbor joining for the whole alignment and optimizes branch lengths and mutation model parameters in a maximum likelihood framework. Then, GARD estimates a tree topology to the left and right of every position p using the same technique as for the global tree and sums the likelihoods for the left and right tree. The likelihood with recombination is based on two topologies with branch lengths and involves $2M$-3 additional parameters compared to the likelihood without recombination. GARD uses the Akaike Information Criterion (AIC) (Hastie, Tibshirani et al. 2001) to penalize for the number of parameters and compares likelihoods involving different numbers of parameters. GARD detects recombination at position p if the AIC with recombination at position p is larger than the AIC for the global tree. Multiple breakpoints are handled analogously: GARD searches for the positions of the breakpoints in the alignment, such that the AIC is maximized. As the number of breakpoint configurations explodes with the number of breakpoints, GARD implements a genetic algorithm to search for the optimal configuration of breakpoints.

GARD implements an interesting strategy as it uses model selection to find the best fitting explanation of the sequences in the alignment. However, GARD introduces too many free parameters for the likelihood with recombination and ignores the correlation between adjacent trees. For example, if there is a single recombination between adjacent trees, the trees are related by one SPR operation and are very similar. One SPR operation only requires three parameters: the source edge in the tree, the destination edge and the position in the destination edge. A major problem of GARD is that it falsely infers recombination if two regions in the alignment evolve with different rates, for example because of a mutation hotspot. In this case, two trees with the same topology but different branch lengths may provide a better fit to the data than a single tree. Rate variation in simulated data does not fully reproduce this effect, as the rates vary randomly across the alignment and do not cluster in regions. GARD does not provide p-values for its output. Nevertheless, a detected recombination is usually significant, as the method is based on a model selection strategy. According to the authors, the performance of GARD is comparable to the best methods in the power study (Posada and Crandall 2001). But GARD may perform worse in practice as the power study does not consider the case of different mutation rates for different parts of the alignment. GARD is only available as a web-service, runs on a compute cluster and has a prohibitive running time if more than a single recombination breakpoint is allowed.

4.3.5.2 Multiple Change Point Model (MCPM)

The MCPM (Suchard, Weiss et al. 2002; Suchard, Weiss et al. 2003) is a Bayesian method for detecting spatial phylogenetic variation (see also PLATO) and for detecting recombination as a special case of spatial phylogenetic variation. The fundamental idea of the MCPM is to partition the alignment at K breakpoints with locations x_k and to allow for a separate topology τ_k, branch lengths b_k and mutation model θ_k in each partition. The overall likelihood of an alignment is then

$$L = \prod_{k=1}^{K+1} \prod_{p=x_{k-1}}^{x_k-1} \Pr\left(A_p \mid \tau_k, b_k, \theta_k\right) \tag{4.9}$$

where x_0=1, x_{K+1}=N+1 and Pr(A_p|τ_k,b_k,θ_k) denotes the probability of the p-th column in the alignment given the parameters. The basic MCPM assumes an uninformative prior on the topologies and does not model that adjacent topologies are correlated by SPR operations in the case of recombination. It is theoretically possible to incorporate a more detailed model of recombination into the prior of the τ_k's and account for SPR operations. The MCPM only considers a rather simple prior and assigns the same prior probability Pr(τ_i|τ_{i-1}) to any topology change. In this case, the MCPM is very closely related to the HMM approach described below. The MCPM specifies priors for all parameters K, x_i, τ_i, b_i and θ_i and then samples the number of breakpoints, the breakpoint locations and the topologies from the posterior using Markov chain Monte Carlo. It is not simple to detect recombination using the output of the MCPM, as the number of inferred breakpoint locations depends on the prior for K. An extension to the MCPM uses a Bayes factor test as a more robust alternative to detect the presence of recombination in the alignment (Suchard, Weiss et al. 2002). The MCPM is very compute intense and is only applicable to few sequences and short alignments. In particular, the number of topologies increases super-exponentially with the number of sequences M (Felsenstein 1978).

There is an interesting extension to the basic MCPM approach that avoids the problem that the topology space grows quickly with the number of sequences (Suchard, Weiss et al. 2002). The idea is to restrict the analysis to the subtyping problem and assume that the topology of the reference sequences is given and fixed. In this case, the MCPM only has to determine where the query sequence attaches to the topology of the reference sequences. In other words, if there are M reference sequences, there are only 2M-3 topologies to consider, as there are only 2M-3 edges to which the query sequence could be attached. This restriction leads to a heavily optimized implementation of the MCPM (Fang, Ding et al. 2006), such that subtyping a 1500nt query sequence regarding eight reference sequences takes about one minute. The output of the MCPM is particularly attractive as it computes the posterior probability that the query sequence is related to each of the reference sequences. This is also a good measure of robustness of the estimate.

There is another extension to the MCPM that is of interest for recombination analysis. The basic MCPM assumes that all parameters τ_i, b_i and θ_i change at the same breakpoint. A breakpoint is therefore not genuinely related to recombination, but could also occur at

the boundaries of a mutation hotspot. To model recombination more accurately, the authors suggest a dual MCPM (Minin, Dorman et al. 2005) that introduces a sequence of breakpoints c_j that is independent of the b_i's. Changes in topology and branch lengths are only allowed at the breakpoints b_i and changes of the mutation model parameters may only occur at the breakpoints c_i. The same sampling approach is applied as before.
The MCPM was not compared to other methods in a power study, but is probably the most accurate model for subtyping and should perform very well in practice. In the case of subtyping, the MCPM models recombination accurately, as there is at most one SPR operation between two sites. But in the more general case, the simple prior does not model recombination accurately. Finally, the MCPM approach is computationally very demanding and can only be applied to short alignments with very few sequences.

4.3.5.3 A Hidden-Markov Model for Computing the Likelihood (HMM)

Husmeier and co-workers introduced a likelihood approach to the analysis of recombinant sequences and considered two different criteria for optimization: maximum likelihood (Husmeier and Wright 2001; 2005) and Bayesian inference (Husmeier and McGuire 2002; Husmeier and McGuire 2003; 2005). Let $\mathbf{A}=(A_1,\ldots,A_N)$ be the alignment with columns A_p, $\boldsymbol{\tau}=(\tau_1,\ldots,\tau_N)$ the unrooted topologies for positions 1 to N and v the probability for recombination. To control the computational complexity, the input is limited to four sequences and there are only three unrooted topologies. Define a set of parameters θ_t representing the branch lengths and the mutation model for each tree topology t. In other words, if there are two positions i and j with the same topology $\tau_i=\tau_j$, the branch lengths and mutation model are the same for both positions. For notational convenience, introduce a vector $\boldsymbol{\theta}=(\theta_1,\theta_2,\theta_3)$ that contains the mutation model and branch lengths for the three tree topologies. The likelihood $\Pr(A_p|\tau_p,\boldsymbol{\theta})$ of a column p in the alignment depends on the tree topology τ_p and the parameters $\boldsymbol{\theta}$. The likelihood of an alignment can then be decomposed as

$$\Pr(\mathbf{A},\boldsymbol{\tau}\,|\,\boldsymbol{\theta},v)=\Pr(\mathbf{A}\,|\,\boldsymbol{\tau},\boldsymbol{\theta},v)P(\boldsymbol{\tau})=\Pr(A_1\,|\,\tau_1,\boldsymbol{\theta})\Pr(\tau_1)\prod_{p=2}^{N}\Pr(A_p\,|\,\tau_p,\boldsymbol{\theta})\Pr(\tau_p\,|\,\tau_{p-1},v) \quad (4.10)$$

where $\Pr(\tau_p|\tau_{p\text{-}1},v)$ models the dependency between adjacent topologies. In theory, $\Pr(\tau_p|\tau_{p\text{-}1},v)$ could model recombination accurately and assign a lower probability to topology changes that involve a larger number of SPR operations. In practice, every topology change is considered equally likely. As $\Pr(\tau_p|\tau_{p\text{-}1},v)$ is only conditional on the neighboring topology $\tau_{p\text{-}1}$, the state sequence $\boldsymbol{\tau}$ can be represented with a hidden Markov model and inference is tractable even for larger alignments. Maximizing $\Pr(\mathbf{A},\boldsymbol{\tau}|\boldsymbol{\theta},v)$ with respect to $\boldsymbol{\tau}$, $\boldsymbol{\theta}$ and v results in the maximum likelihood estimate of the local trees $\boldsymbol{\tau}$ and the recombination probability v. The actual optimization is carried out with an expectation-maximization algorithm, where the E-step estimates the probability of the topologies $\boldsymbol{\tau}$ given the other parameters and the M-step maximizes the likelihood of the other parameters $\boldsymbol{\theta}$ and v. The Bayesian approach uses Markov chain Monte Carlo (MCMC) to sample

topology sequences $\boldsymbol{\tau}$ from the posterior. The Bayesian approach is implemented in TOPALi (Milne, Wright et al. 2004) and is referred to as HMM analysis in the following. The result of an HMM analysis can be visualized by plotting the posterior distribution for each topology along the sequence positions. Recombination is detected if the posterior suggests different topologies for different positions in the alignment.
The HMM analysis is similar to the MCPM approach, but models transitions between topologies in more detail. Nevertheless, recombination is not correctly modeled as an SPR operation on rooted, ordered binary trees, and the parameters $\boldsymbol{\theta}$ are also fixed for each possible topology. The restriction to only four input sequences is very limiting in a practical setting. It is difficult to select four sequences from the alignment such that the recombination signal is clearly shown in the analysis. An unknown alignment requires extensive manual interaction and a lot of experience to consolidate the information from several runs with different sequence subsets. As a result, the HMM method is more interesting from a theoretical point of view than it is for practical use.

4.3.6 Methods That Infer a Restricted ARG

There are several methods that reconstruct a restricted version of the ARG and avoid the NP-complete recombination parsimony problem. Nevertheless, some approaches still have a very high computational demand.

4.3.6.1 RecPars

Hein (Hein 1990) not only described the recombination parsimony problem (see section 4.2.7), but also introduced several algorithms to arrive at an approximate solution. The first approximation is based on the fact that the number of rooted, ordered binary tree topologies grows much faster than the number of unrooted tree topologies. Therefore and since inferring the root position is usually a difficult problem, only unrooted topologies ideally would be considered in the algorithm. However, every SPR operation on an unrooted topology introduces constraints on the possible position of the root, and this information has to be carried over as a restriction for subsequent SPR operations. Hence, Hein (Hein 1990) described how to restrict the search algorithm to root restricted topologies, a class of topologies between unrooted and fully rooted topologies. Interestingly, Hein (Hein 1993) later found out that this algorithm does not accurately solve the recombination parsimony problem. Root restricted trees ignore the ordering of nodes and the SPR operations may not represent recombination correctly. We cannot avoid considering all rooted, ordered binary tree topologies if recombination is modeled correctly.
Based on the formulation of recombination parsimony as a dynamic program, Hein (Hein 1993) developed a heuristic algorithm called RecPars. RecPars makes several simplifying assumptions. First, for each site at most one recombination is allowed that has a breakpoint just right to the site, so that neighboring trees only differ by a single SPR operation. Second, the correct tree topology for the first site should be known. Finally, only un-

rooted topologies are considered. The relative ordering of coalescent events is ignored and it might be impossible to interpret an SPR operation as a single recombination event. RecPars then determines the topologies for each site by scanning the alignment from left to right. Given the correct topology at the first site, RecPars considers all topologies that are one SPR operation away for the second site. As the dynamic program progresses towards the last site, it may become optimal to switch to another topology. If this happens, all topologies that are one SPR operation away from the newly optimal topology are generated and the dynamic program only continues to scan these topologies for optimality for the following sites. As soon as the alignment is fully processed, the topology for each position can be recovered and the breakpoints can be estimated, too. Hein (Hein 1993) also describes a heuristic for optimizing the initial tree. The tree for the full alignment, or alternatively for the first couple of sites, is reconstructed as a preliminary estimate for the initial tree. Then, RecPars performs a full scan from left to right, determining the trees for each site and in particular for the last site. The hope is that the optimized tree for the last site does not depend much on the potentially wrong estimate of the initial tree and should therefore be more accurate than the initial tree estimate. Using the tree for the last position as input, RecPars then performs a final run from right to left and recovers the optimal solution.

There are several aspects that may cause RecPars to find a suboptimal solution, even if there is at most one recombination per site. First, restricting the analysis to unrooted topologies misses recombination events that do not change the unrooted topology, but cause a change of the rooted, binary topology. Fortunately, this does not falsely infer recombination events and only reduces the power of detecting recombination. Second, a wrong initial tree may affect the solution of RecPars, but the details of this effect are unclear. Finally, if there are multiple topologies that involve the same cost, RecPars only greedily follows a single topology and may miss the optimal path through the tableau.

The work of Hein (Hein 1990; Hein 1993) is groundbreaking in many respects. It introduces the fundamental recombination parsimony problem and also describes many details of recombinations as SPR operations. However, the complete definition of a recombination as an SPR operation on rooted, ordered binary trees was developed later in (Song and Hein 2003; Song 2006). In comparison to other approaches, Hein models the correlation between adjacent trees accurately and does not rely on the vague criterion of different trees for different regions. RecPars is also one of the few methods that attempt inferring the full history of a set of sequences. Even though the basic approach is very promising, RecPars is limited by the assumptions it introduces. RecPars can only detect recombination in unrooted topologies and is therefore restricted to alignments with more than three sequences. The solution that RecPars proposes also depends heavily on the cost for recombination. It can be difficult to judge whether the recombination events RecPars infers are correct or if they are the result of recurrent mutations and a low cost associated with recombination. The column permutation test can be used to compute the *p*-value for re-

combination in the alignment, for example by comparing the inferred number of recombination events in the original and in the permuted alignments (Posada and Crandall 2001). But it is actually not clear how to derive *p*-values for other parts of the output of RecPars. Power studies (Posada and Crandall 2001; Wiuf, Christensen et al. 2001) have shown that the *p*-value derived from RecPars often performs worse for detecting recombination than simple methods, such as the MC^2 method or NSS. In special scenarios, such as exponential growth, RecPars performs better than simple methods.

4.3.6.2 Minimal Ancestral Recombination Graph (MARG)

The MARG method (Song and Hein 2005) models recombination accurately as an SPR operation on rooted, ordered binary trees. In comparison to RecPars, it does not only model recombination on unrooted topologies, allows for more than one recombination event with the same breakpoint position and does not require a correct topology for the first site as an input. In other words, the MARG method attempts to solve the recombination parsimony problem accurately. Unfortunately, it is even difficult to compute the minimum number of SPR operations for transforming a rooted, ordered binary tree into another. A simple solution is to precompute all trees that are one recombination away for every rooted, ordered binary tree and use the stored results later in the algorithm. Interestingly, this representation works particularly well for the infinite sites model, because the recursion equations of the dynamic program in (4.2) can be nicely adapted to it. Alignments of about 10 sequences can be processed in reasonable time in this case. It is possible to extend the algorithm to the finite sites model, but the recursion equations involve a large state space (see section 4.2.7) and require a prohibitive running time.

The MARG method provides many insights into the combinatorial structure of the recombination parsimony problem, particularly with respect to SPR operations on rooted, ordered binary trees. As far as the recombination analysis of real world sequences is concerned, it is unclear if the assumptions introduced by RecPars are more deteriorating than the infinite sites assumption of the MARG method. The validation of the MARG method only considered evolutionary scenarios that satisfy the infinite sites assumption and compared it to several other methods estimating the minimum number of recombination events.

4.3.7 Estimators of the Recombination Rate ρ

Several estimators of the scaled recombination rate ρ have been developed based on the coalescent framework. Estimators of the recombination rate are concerned with properties of the population, such as the overall recombination rate or the local variation of the recombination rate. They usually do not attempt to analyze the recombination signal in more detail and estimate the recombination structure of a single sequence, for example. Estimates of ρ are very useful for studies that map disease genes, but they can also be

used to detect recombination in the alignment or the locations of recombination hotspots (Stumpf and McVean 2003).

There are three different categories of estimators (Stumpf and McVean 2003). Moment-based estimators use simple summary statistics to estimate ρ and are usually fast, but not very accurate. Approximate likelihood estimators compute the likelihood of ρ for the whole alignment by summing the likelihood of ρ for parts of the alignment (Smith and Fearnhead 2005). For example, Hudson's method (Hudson 2001) sums the likelihood of ρ for every pair of sites i and j to obtain the likelihood estimate of ρ for the whole alignment. Finally, full-likelihood approaches compute the probability distribution of ρ given the alignment and are very accurate, but extremely slow.

The full-likelihood approaches are of particular interest as, in principle, they can estimate other quantities as well. RECOMBINE (Kuhner, Yamato et al. 2000) samples a set of ARGs from the posterior probability

$$\Pr(A \mid G)\Pr(G \mid \rho_0, \theta_0) \tag{4.11}$$

where $\Pr(A|G)$ is the probability of the alignment given an ARG G and $\Pr(G|\rho_0,\theta_0)$ denotes the coalescent probability of the genealogy G depending on some starting parameters ρ_0 and θ_0. $\Pr(A|G)$ can be computed by methods known from maximum likelihood phylogenetic tree inference. Given the sampled ARGs, RECOMBINE then approximates the joint likelihood distribution $\Pr(A|\rho,\theta)$ for the population genetic parameters ρ and θ and computes their maximum likelihood estimate. The estimate is biased by the starting parameters ρ_0 and θ_0, such that the method should be rerun several times with the maximum likelihood estimates as the input. Interestingly, it is also possible to approximate the likelihood distribution of other statistics that depend on the sampled ARGs, for example, the probability for recombination between every site.

RECOMBINE can also be used for detecting recombination in an alignment (Brown, Garner et al. 2001). Unfortunately, it is difficult to infer that the recombination rate is zero, as there might have been invisible recombination events in the alignment. An approach to avoid this difficulty is to use a likelihood ratio test:

$$LRT = L(A \mid \rho = 0) \Big/ \max_{\rho} \{L(A \mid \rho)\} \tag{4.12}$$

Recombination is then detected if the likelihood ratio test statistic LRT is significantly large than the LRT for alignments simulated without recombination. Brown and coworkers conducted a power study with a protocol that has a small bias in favor of the likelihood ratio test. The power study showed that the likelihood ratio test performs much better than some of the best classical methods, such as Geneconv or NSS. Hence, there is still room for further improvement in recombination detection.

Even though full-likelihood estimators are a promising approach for the analysis of recombination, there are several problems that often prevent their application in practice. The user usually has to correctly set several parameters, such that the method samples enough genealogies and the estimate converges. Unfortunately, it is difficult to assess whether the method already sampled enough genealogies. This is not a problem specific

to RECOMBINE and affects other approaches as well (Fearnhead and Donnelly 2001). The running time of the full-likelihood estimators are usually in the order of hours or even days for alignments with 10,000 sites and 50 sequences (Fearnhead and Donnelly 2001). Finally, full-likelihood estimators only approximate the likelihood of the recombination rate ρ or the recombination rate at every site. As the estimated quantities become more complex, RECOMBINE has to sample a lot more genealogies to arrive at a good estimate of the likelihood.

4.3.8 Other Methods and Problems

There are many publications that propose an optimization problem related to the analysis of recombination and then analyze the complexity of the problem or develop algorithms for finding an optimal or near-optimal solution. However, it is often difficult to judge whether the optimization problem is relevant for the analysis of recombination, particularly as the algorithms are often not implemented and usually not validated on realistic datasets. Important questions for an application in practice are often ignored, for example measuring the robustness of the output. Still, these publications can be a great resource for finding problem settings related to recombination parsimony, which are solvable in polynomial time or have other desirable properties.

4.3.8.1 Methods Related to Recco

The next chapter introduces Recco, the method developed in this thesis. Several optimization problems in literature are related to Recco, but do not fit the definition of recombination analysis as described before.

Kececioglu and Gusfield (Kececioglu and Gusfield 1998) study the problem of reconstructing a spanning history of a set of sequences allowing for mutation and recombination. The input can be an arbitrary set of sequences and does not require a multiple alignment. The history of a set of sequences is modeled by a path in a directed hypergraph. Each edge in the hypergraph connects three nodes: two parental nodes and the child node. The sequence of the child node can be derived by recombination from the parents. The problem is then to find the pair of sequences in the dataset and a path in the hypergraph such that all sequences in the dataset are reconstructed with minimum cost. Even though this problem is NP-complete in general, it can be solved in polynomial time for restricted models of the recombination operator. The authors also introduce a dynamic algorithm that reconstructs a child sequence with minimum cost from the two parental sequences, allowing for mutation and unequal recombination. As described earlier, the effect of unequal recombination is similar to allowing for insertions and deletions. The complexity of the algorithm for the most general case is $O(N^3)$.

Lajoie and El-Mabrouk (Lajoie and El-Mabrouk 2005) use a multiple alignment as input and derive a query sequence in the alignment from the other sequences, allowing for mutation, recombination and gene conversion. As for the jumping alignments method, every

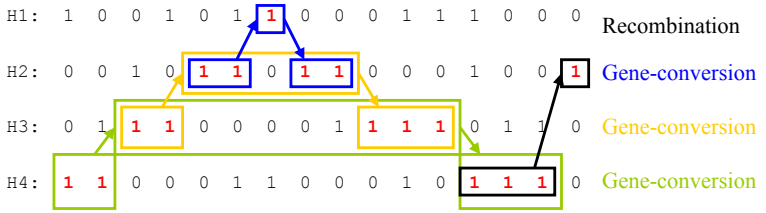


Figure 4.6: The query sequence consists of all 1's and is not shown in the multiple alignment. The method (Lajoie and El-Mabrouk 2005) then explains the query sequence by a path from left to right through the alignment. The path is visualized by the 1's that are colored red. Each transition to another sequence is either caused by a gene conversion or by a recombination event. Gene-conversion event 1 integrates part of sequence H3 into H4. The resulting sequence is then subject to gene-conversion event 2 and so on. Irrelevant parts for the gene-conversion events are not enclosed in a colored box for simplicity. The example is adopted from (Lajoie and El-Mabrouk 2005).

position of the query sequence is derived from a single sequence in the alignment. Consequently, the query sequence can also be described by a path from left to right through the multiple alignment (see Figure 4.6). The optimal solution is computed by dynamic programming in time $O(NMS^2)$, where S is the maximum length of a segment exchanged by a gene conversion event.

4.3.8.2 Bounds on the Minimum Number of Recombination Events

Suppose that the sequences in the input alignment evolved according to the infinite sites model. As there are no recurrent mutations in this case, each local tree fits the mutation pattern perfectly and the total number of mutations is equal to the number of polymorphic sites. Thus, the recombination parsimony problem (see section 4.2.7) does not have to penalize for mutations, but only has to minimize the number of recombinations in the inferred history. An important problem in this setting is to infer the minimum number of recombination events R_{min} necessary to explain an alignment. Unfortunately, even this problem is very difficult. A variation of the problem where the sequence of the MRCA is given, is claimed to be NP-hard (Wang, Zhang et al. 2001; Song, Wu et al. 2005). One approach for computing R_{min} is the MARG method (see above). There are several other methods that provide lower bounds for R_{min} (Myers and Griffiths 2003; Bafna and Bansal 2004; Song, Wu et al. 2005), but even newer methods often need minutes to process alignments with less than 1000 sites and 25 sequences if an accurate bound is desired (Song, Wu et al. 2005).

The column permutation test can be used to compute a p-value for recombination in the alignment based on the approximate value R_{min}. But the high computational demand

usually forbids such an approach. Nevertheless, a comparison of these methods with other recombination detection programs would be of general interest, as the impact of the infinite sites assumption could be assessed. However, the field is mainly concerned with a better and faster estimate of R_{min}, and validation is usually restricted to alignments simulated according to the infinite sites model.

4.4 Comparison of Methods for the Analysis of Recombination

Table 4.1 lists most methods of the previous section and compares them regarding accuracy, speed and output. Most comparisons in the table are qualitative and approximate. The reader should refer to the corresponding section for more details on each method. The table also lists Recco, the method developed later in this thesis. It is usually difficult to compare the accuracy of the methods objectively if the method is not evaluated in one of the power studies (Posada and Crandall 2001; Wiuf, Christensen et al. 2001). Moreover, the comparison in the power studies only refers to the power of detecting recombination, but does not evaluate the accuracy of more detailed predictions, such as predicting the recombinant sequence or the breakpoint. The power study presented in section 7.1 is more detailed and provides a comparison of Recco with Geneconv, NSS, PHI, RecPars and the MC^2 method regarding the power of detecting the recombinant sequence, too. The table also does not convey if the output of a method is easy to interpret. Finally, the table lists whether it is possible to treat gaps as missing data, but the reasoning is developed later in chapter 6. Some methods such as TOPAL also do not implement a sensible gap model even though they could do so in principle.

The most important criterion for comparing methods is accuracy. It is surprising that simple methods based on the distribution of mutations (section 4.3.1) or on compatibility (section 4.3.2) perform consistently well for detecting recombination in an alignment. They are also very fast and do not depend on any important input parameter such as the window size. Unfortunately, these methods do not easily allow for a more detailed analysis of the recombination signal. Geneconv and MC^2 only detect recombination in a sequence pair, but not in a single sequence and also do not estimate the structure of a recombinant sequence. Methods that optimize breakpoints and compare local trees (section 4.3.5) may perform similarly or better than the simple methods. However, they require much more computation time and are usually not validated on a larger dataset. The output of these methods is a sequence of unrelated trees and is usually difficult to interpret. An exception is the MCPM in the case of subtyping, as it has been adapted specifically for subtyping.

Few methods enable inferring the structure of recombinant sequences or can solve the more restricted subtyping problem (section 4.3.3). Most methods for subtyping (section 4.3.3) implement very simple strategies that probably do not perform well in practice. The jpHMM is an exception from the rule, but it does not compute *p*-values and is not suitable for recombination analysis in general. Methods that compare local trees and use a

sliding window (section 4.3.4) allow for a more detailed analysis of the recombination signal and are therefore very popular. Unfortunately, they do not perform well for detecting recombination in an alignment and probably are also inaccurate regarding the other predictions. The window size is also critical, as it selects between resolution and robustness of the inferred breakpoint locations.

Methods that try to infer the ARG by maximum parsimony (section 4.3.6) are particularly interesting, as they model the ARG quite accurately. The output is a sequence of trees related by SPR operations and in principle allows addressing all questions in recombination analysis. The recombination parsimony problem would be an interesting gold standard for recombination analysis if it could be solved in reasonable time. As this is not possible, the methods introduce several simplifying assumptions. The MARG method falsely infers recombination events if there are recurrent mutations and is probably inappropriate for real alignments with a lot of mutation. RecPars only infers SPR operations on unrooted topologies and may return a history of recombination and mutation events that cannot be represented as an ARG. The approximation of RecPars is conservative and does not falsely infer recombination events. RecPars does not perform as well for detecting recombination in an alignment, but its false detection rate is also often very much below 0.05.

An important criterion for practical use is also that an implementation is readily available and easy to use. Besides the web-based methods for subtyping, there are several standalone programs. The website of Robertson (http://www.bioinf.manchester.ac.uk/recombination/programs.shtml) is an excellent resource for finding recombination detection programs. Geneconv is a robust and extremely fast command-line program that is ideal for a comparison method in power studies. Likewise, PHI is also very fast and implements the PHI method, the MC^2 method and NSS. RecPars was also implemented as a command-line program (http://www.daimi.au.dk/~compbio/recpars/recpars.html) and is also quite fast. RDP (Martin and Rybicki 2000; Martin, Williamson et al. 2005) provides a very comprehensive analysis tool and combines many different methods in a single GUI, for example the MC^2 method, Geneconv, PhylPro, Bootscanning and TOPAL. Unfortunately, RDP is still beta and crashes quite often. Simplot (Lole, Bollinger et al. 1999) is a very popular program for recombination analysis and computes and visualizes a Bootscanning analysis of an alignment. TOPALi (Milne, Wright et al. 2004) robustly implements the methods TOPAL, PDM and HMM and also provides a simple GUI. Other implementations are less relevant for the analysis of recombination or simply do not work.

4.5 Conclusion

In comparison with tree-based phylogenetic analysis procedures, procedures for analyzing recombination are immature. Recent power studies on recombination detection methods uncovered that it can be hard even to decide whether there is recombination in a set

of aligned sequences (Posada and Crandall 2001; Wiuf, Christensen et al. 2001) and that it is still possible to improve the accuracy of current methods (Brown, Garner et al. 2001). A more detailed analysis of the recombination signal is even more challenging and includes detecting the recombinant sequences, the recombination breakpoints or estimating the structure of a recombinant sequence. Only few methods perform an analysis of recombination in this detail. But they do not perform well for recombination detection, and it is questionable whether their other predictions are more accurate. The best methods for detecting recombination in the alignment scan the alignment for an unequal distribution of mutations or incompatible sites, but cannot estimate the structure of a recombinant sequence. These methods also do not model recombination accurately. It is often difficult to interpret the output of the method and extract recombination and mutation events. As the methods only search for indirect evidence of recombination, such as a change of the tree topology across the alignment, they may also falsely detect recombination for specific evolutionary scenarios.

There are several methods that model recombination more accurately and infer a limited version of the ARG, for example based on parsimony or maximum likelihood. As the underlying problem is intractable for all but the smallest alignments, the methods have to define a restricted version of the problem. The corresponding model assumptions are usually much more transparent than for methods that do not model recombination directly, and the resulting limitations of the method are more clearly defined. It is often possible to interpret the output as an ARG and extract many details of the inferred recombination and mutation events.

	Methods (in section 4.3.x)	Accuracy	Speed (min)	Window-based	Requires topology changes	Gaps	Detects recombination for the			Estimates		Comments
							Alignment	Sequence	Breakpoint	Recombinant structure	ARG	
	Recco	+	0.3[§]	no	no	-	p	p	p	o	-	the method developed in this thesis
1.	**MC²**	+	<0.01[§]	no	no	+	p	(p)[‡]	o	-	-	detects only pairs of rec. seqs
1.	**Geneconv**	+	<0.01[§]	no	no	-	p	(p)[‡]	o	-	-	detects only pairs of rec. seqs
1.	**PhylPro**	+	<0.01[§]	yes	no	+	o/p	o/p	o/p	-	-	output is difficult to interpret without *p*-values
2.	**PHI**	+	<0.01[§]	yes	yes	+	p	-	-	-	-	robust even to population growth
2.	**NSS**	+	<0.01[§]	no	yes	+	p	-	-	-	-	
3.	**RIP**	(-)	<0.01[§]	yes	no	-	o/p	o/p	o/p	o	-	
3.	**STAR**	(--)	?	no	no	?	o	o	-	o	-	
3.	**jpHMM**	?	?	no	no	NA	o	o	o	o	-	needs to estimate an HMM on training data
4.	**Bootscanning**	-	1.1	yes	yes	+	s	s	o	o	-	very popular method
4.	**SlidingBayes**	?	?	yes	yes	+	s	s	o	o	-	program does not run
4.	**PLATO**	--	?	yes	yes	+	s	-	o	-	-	local rate variation leads to wrong results
4.	**TOPAL**	(o)	3	yes	no	+	p	?	p	-	-	
4.	**PDM**	(o)	100	yes	yes	+	s	-	s	-	-	
5.	**GARD**	(+)	0.1; 12[†]	no	yes	+	s	-	s	o	local trees	local rate variation leads to wrong results
5.	**MCPM**	(+)	?	no	no	+	s	?	o	o	local trees	
5.	**HMM**	(+)	0.5	no	no	+	?	?	o	o	local trees	only analyzes four sequences at a time
6.	**RecPars**	o	0.5	no	yes	+	o/p	(-)	o	o	(o)	uses a nice heuristic for ARG inference
6.	**MARG**	?	?	no	no	+	o/p	o	o	o	o	assumes the infinite sites model

[§] computation time includes 100 column permutations for computing *p*-values
[†] runs remotely on a compute cluster; first number is for optimizing a single breakpoint only
[‡] *p*-value only for recombination in a sequence pair

Table 4.1: Methods for recombination analysis are compared regarding their basic properties. **Accuracy:** The accuracy is based on the power studies (Posada and Crandall 2001; Wiuf, Christensen et al. 2001) or on the original publication if it is comparable to one of the power studies. Values in brackets indicate personal judgment and '?' indicates missing data. **Speed:** Shown is the time required for an alignment of 10 sequences and 1000 nucleotides and typical settings for the method. **Window-based:** Using a sliding window has several negative consequences. **Topology change:** Several methods can only detect recombination if the unrooted topology of the local coalescent tree changes. **Gaps:** Methods are identified by '+' if, in principle, they can handle gaps as missing data. **Detects recombination for:** The method may detect any recombination in the alignment, the recombinant sequence or the recombination breakpoint. 'o' stands for a point estimate, 'p' indicates that a measure of robustness is provided, and 's' is used if the method generates significant solutions without a measure of robustness. 'o/p' indicates that it is simple to derive *p*-values with the column permutation test, but that the method does not implement it. **Estimates:** Some methods estimate the recombinant structure of a sequence or the ARG for the alignment. 'local trees' indicates that the method finds breakpoints and estimates a (possibly unrelated) set of trees for the intervals, but does not model recombination as a SPR-operation.

5 Recco – Recombination Analysis Through Cost Optimization

The following chapter introduces Recco (Maydt and Lengauer 2006), the method for recombination analysis developed in this thesis. The fundamental idea of Recco is to reconstruct one sequence in the alignment from the other sequences using mutation and recombination, such that the sum of costs for mutation and recombination is minimized. The optimization problem can be solved efficiently by dynamic programming and introduces parameters for the mutation and recombination cost (see section 5.2 and 5.3). In practice, a single parameter α representing the cost of mutation against recombination cost is usually sufficient. The minimum-cost solution identifies the best recombination breakpoints and the parental sequences for the query sequence. But the minimum-cost solution heavily depends on the parameter α and does not provide a measure of robustness. Recco addresses these problems by two types of sensitivity analysis for the optimization problem (see section 5.4). The first analysis studies the minimal cost of configurations that are close to the minimum-cost solution regarding their configuration and identifies critical and robust parts of the minimum-cost solution for a fixed α. The second analysis describes a parametric analysis regarding α, such that all pareto-optimal solutions can be recovered. Particularly the second analysis is useful in practice and allows for computing the Savings value, an intuitive indicator of recombination in the query sequence. The Savings value measures how many mutations can be eliminated from the solution by introducing one additional recombination. Section 5.5 then draws the connection to the ARG and describes the Savings value and the pareto-optimal solutions that we would expect for an alignment simulated according to a known ARG. Section 5.6 brings the different parts together and describes the full analysis performed by Recco. Recco computes p-values for recombination in the alignment, in a sequence and for a breakpoint using the column permutation test. But Recco also introduces the concept of a recombination event list and determines a cutoff, such that a robust hypothesis for the query sequence is extracted. Section 5.6 also describes a simple heuristic that avoids an inherent limitation of the Recco analysis: the basic approach cannot identify a recombination event for a sequence if there is a highly similar sequence contained in the alignment. This limitation is not critical for a manual analysis, but has a negative impact on the fully automated recombination detection performance. Finally, section 5.7 discusses several options for improving the speed of the Recco analysis.

5.1 An Example Alignment

In order to illustrate Recco, I refer to the same sequence alignment throughout the discussion. The alignment was generated by the program Seq-Gen (Rambaut and Grassly 1997) using the genealogy depicted in Figure 5.1. Sequences R1 and R2 are created by recombining the ancestor of A1 to the left of the breakpoint with the ancestor of B1 to the right

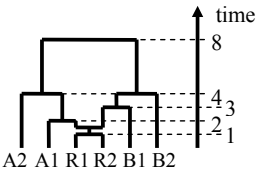


Figure 5.1: The genealogy used for generating the example alignment. See the text for details.

of the breakpoint and only differ by mutations that occurred after the recombination event. The breakpoint occurs in the middle of the sequence after the 50th nucleotide and all sequences have a length of 100 nucleotides. Nucleotide substitution was simulated with the Jukes-Cantor model of nucleotide evolution (Jukes and Cantor 1969) (see section 3.2) such that there are on average 0.25 substitutions per site from the root to any of the tips of the genealogy. The alignment is shown several times throughout this chapter, for example in Figure 5.6. The alignment poses a particular problem for Recco, as both recombinants R1 and R2 are very similar. Hence, the discussion usually refers to the alignment without sequence R2 for the purpose of the exposition. The full alignment including sequence R2 is analyzed in section 5.6.6.

5.2 The Basic Optimization Problem

Recco takes a multiple alignment as input and uses one sequence of the multiple alignment as the query sequence. In the following, assume that the query sequence is removed from the multiple alignment, as this representation is more suitable for the mathematical presentation. To be more specific, N is the number of sites in the alignment and M is the number of sequences excluding the query sequence. The multiple alignment excluding the query sequence is represented by a matrix $\mathbf{A}$ where A_{ip} is the p-th nucleotide of the i-th sequence and A_i represents the i-th sequence of the alignment. Let $\mathbf{s}=s_1,\ldots,s_N$ be the nucleotide sequence of the query. Let $m_{\mathbf{A},\mathbf{s}}(i,p)\geq 0$ be the cost of substituting A_{ip} with s_p. As many functions depend on $\mathbf{A}$ and $\mathbf{s}$ in the following, these dependencies are omitted and $m_{\mathbf{A},\mathbf{s}}(i,p)$ is referred to as $m(i,p)$. A common choice in practice is to use Hamming distances and set $m(i,p)=\delta(A_{ip},s_p)$ where $\delta(x,y)$ is defined as $\delta(x,y)=0$ if $x=y$ and $\delta(x,y)=1$ otherwise. Recombination cost is measured by $r(i,j,p)\geq 0$ and represents the cost of recombining A_i on the left side of position p with A_j on the right side, hence resulting in the sequence $A_{i,1},\ldots,A_{i,p},A_{j,p+1},\ldots,A_{j,N}$. Note that this definition is different from the publication of Recco (Maydt and Lengauer 2006), as $r(i,j,1)$ refers to a recombination *after* the first nucleotide. It makes sense to set $r(i,i,p)=0$ for all i and p, as this does not represent a visible recombination event. We usually also assume for the theoretical analysis that the recombination cost does not depend on the donor and acceptor sequences A_i and A_j, but only on the position p. In other words, $r(i,j,p)=\delta(i,j)r(p)$, where $r(p)$ is the cost of a recom-

bination at position p. All empirical analyses use an even simpler setting, the so-called Hamming distances for recombination cost: $r(i,j,p)=\delta(i,j)$.

The goal is to find a series of mutation and recombination events that produce the query sequence from the sequences in **A** with minimum cost. A more suitable representation of the series of events in our case is a path $\mathbf{u}=(u_1,\ldots,u_N)$ through the alignment, where $u_p \in \{1,\ldots,M\}$ for all p determines which sequence of the alignment is used to explain position p of the query sequence. In other words, a path **u** defines a sequence $\mathbf{s'} = A_{u_1,1},\ldots,A_{u_N,N}$ composed of the sequences in A and introduces a mutation event if $A_{u_p,p} \neq s_p$. The total cost of mutation from **s** to **s'** is then $M(\mathbf{u},1,N)$, where

$$M(\mathbf{u},x,y)=\sum_{p=x}^{y} m(u_p,p) \tag{5.1}$$

is the cost of mutation in the interval $[x,y]$. The path **u** also introduces recombination events at, or more exactly directly after the positions $\Delta(\mathbf{u})=\{p|u_p \neq u_{p+1}\}$ in the alignment. The recombination cost associated with a path **u** is defined as $R(\mathbf{u},1,N)$, where $R(\mathbf{u},x,y)$ is the recombination cost in the interval $[x,y]$:

$$R(\mathbf{u},x,y)=\sum_{p=x}^{y-1} r(u_p,u_{p+1},p)\,. \tag{5.2}$$

The total cost of a path **u** is $\varphi_\alpha(\mathbf{u},1,N)$, where $\varphi_\alpha(\mathbf{u},x,y)$ is the total cost in the interval $[x,y]$ and the parameter $\alpha \in [0,1]$ weighs mutation cost against recombination cost:

$$\varphi_\alpha(\mathbf{u},x,y)=(1-\alpha)R(\mathbf{u},x,y)+aM(\mathbf{u},x,y)\,. \tag{5.3}$$

A path **u** is called α-optimal or an α-optimal solution if it minimizes $\varphi_\alpha(\mathbf{u},1,N)$ for a given α and therefore if $\varphi_\alpha(\mathbf{u},1,N) \leq \varphi_\alpha(\mathbf{v},1,N)$ for all paths **v**. The basic optimization problem then finds all α-optimal paths for a given α:

$$\arg\min_{\mathbf{u}} \varphi_\alpha(\mathbf{u},1,N)=\arg\min_{\mathbf{u}}\left\{(1-\alpha)\sum_{p=1}^{N-1} r(u_p,u_{p+1},p)+\alpha\sum_{p=1}^{N} m(u_p,p)\right\}. \tag{5.4}$$

The parameter α accounts for the fact that a recombination event can always be substituted by mutation events. If α is close to zero, the α-optimal paths favor mutations. As α converges towards zero, the α-optimal path eventually explains the query sequence only by mutation from the closest sequence i. For α=1 recombination costs nothing and each α-optimal path uses a nucleotide A_{ip} that is most similar to s_p at each position p.

If we use Hamming distances for recombination cost and mutation cost, each recombination costs as much as $(1-\alpha)/\alpha$ mutations. For example, each recombination costs as much as four mutations if α=0.2. In other words, α quantifies the global preference for recombination against mutation. Hence, the mutation cost $m(i,p)$ and the recombination cost $r(i,j,p)$ can be designed and normalized independently from each other. They model prior knowledge on the recombination and mutation process, such as local changes in recombination and mutation frequency. A mutation hotspot or a position p in a variable region might have a lower cost $m(i,p)$ for substitution than other positions. Also, mutation to a rare nucleotide can incur a higher cost than mutation to a common nucleotide, thereby

accommodating nucleotide composition bias. There are many publications on methods estimating mutation rate heterogeneity (Pesole and Saccone 2001; Meyer and von Haeseler 2003) or global mutation rate matrices (Altschul 1991; Henikoff and Henikoff 1992), although some of these assume a tree-like evolutionary history and may not directly apply to recombinant sequences. But in principle, the resulting information is easily incorporated into the mutation cost.

The recombination cost parameter $r(i,j,p)$ can also vary among positions and capture the lower cost at recombination hotspots. However, it is also possible to model much more complex effects, as $r(i,j,p)$ also depends on the sequence pair (i,j). For example, several mechanisms proposed for HIV recombination (see section 2.1.3.1) depend on the donor sequence i, on the acceptor sequence j or on the pair of parental sequences (i,j) that produce the recombinant. Such effects can easily be accounted for by increasing or decreasing the respective recombination cost $r(i,j,p)$ in a preprocessing step. But as the prevalent recombination mechanism is unknown, it is more conservative to use Hamming distances for $r(i,j,p)$ or a similarly simple setting. Imposing a structure on $r(i,j,p)$ affects the solution and complicates its interpretation considerably. For example, the solution favors recombinations at positions p where $r(p)$ is small. Furthermore, all proofs in this chapter depend on the assumption that $r(i,j,p)=\delta(i,j)r(p)$ or even $r(i,j,p)=\delta(i,j)$, as the structure of the optimal path is much more complex otherwise and does not allow for an analytical result. It is also possible to develop a faster solver for the optimization problem if $r(i,j,p)=\delta(i,j)r(p)$. Finally, all empirical analyses in this thesis use the simple settings of $m(i,p)=\delta(A_{ip},s_p)$ and $r(i,j,p)=\delta(i,j)$.

5.3 Formulation as a Dynamic Program

Several additional definitions are used in the following. A path $\mathbf{u}$ is (α,p)-prefix optimal if all paths $\mathbf{v}$ with $v_p=u_p$ satisfy $\varphi_\alpha(\mathbf{u},1,p)\leq\varphi_\alpha(\mathbf{v},1,p)$. In other words, there is no cheaper path $\mathbf{v}$ for positions 1 to p ending in sequence u_p. The cost of an (α,p)-prefix optimal path ending in sequence u_p is therefore

$$f_{u_p,p} = \min_{u_1,\ldots,u_{p-1}} \{\varphi_\alpha(\mathbf{u},1,p)\}. \tag{5.5}$$

A path $\mathbf{u}$ is (α,p)-suffix optimal if all paths $\mathbf{v}$ with $v_p=u_p$ satisfy $\varphi_\alpha(\mathbf{u},p,N)\leq\varphi_\alpha(\mathbf{v},p,N)$. The cost of an (α,p)-suffix optimal path ending in sequence u_p is

$$b_{u_p,p} = \min_{u_{p+1},\ldots,u_N} \{\varphi_\alpha(\mathbf{u},p,N)\}. \tag{5.6}$$

Clearly, an (α,p)-prefix optimal path is $(\alpha,p\text{-}1)$-prefix optimal and an (α,p)-suffix optimal path is $(\alpha,p+1)$-suffix optimal. It also follows directly from the definitions above that an α-optimal path $\mathbf{u}$ is (α,N)-prefix optimal and $(\alpha,1)$-suffix optimal. However, an (α,N)-prefix optimal path is not necessarily α-optimal, as there might be a path that does not end in sequence u_N and involves a lower cost. An (α,N)-prefix optimal path $\mathbf{u}$ ending in sequence u_N is α-optimal if and only if there is no other (α,N)-prefix optimal path $\mathbf{v}$

with $\varphi_\alpha(\mathbf{v},1,N)<\varphi_\alpha(\mathbf{u},1,N)$. These observations carry over to suffix optimality in a similar way.

The proposed optimization problem (5.4) can be solved efficiently by dynamic programming, as the cost function is separable (Domschke and Drexl 1998):

Observation "Separability of the Cost Function":

The cost function φ_α is separable into $\varphi_\alpha(\mathbf{u},x,y)=\varphi_\alpha(\mathbf{u},x,z)+\varphi_\alpha(\mathbf{u},z+1,y)+(1-\alpha)r(u_z,u_{z+1},z)$ for any x,y,z with $x\le z<y$.

Proof:

The recombination and mutation cost functions can be decomposed for any x,y,z with $x\le z<y$:

$$\begin{aligned} M(\mathbf{u},x,y) &= \sum_{p=x}^{y} m(u_p,p) = \sum_{p=x}^{z} m(u_p,p) + \sum_{p=z+1}^{y} m(u_p,p) \\ &= M(\mathbf{u},x,z) + M(\mathbf{u},z+1,y) \\ R(\mathbf{u},x,y) &= \sum_{p=x}^{y-1} r(u_p,u_{p+1},p) = \sum_{p=x}^{z-1} r(u_p,u_{p+1},p) + r(u_{z-1},u_z,p) + \sum_{p=z+1}^{y-1} r(u_p,u_{p+1},p) \\ &= R(\mathbf{u},x,z) + R(\mathbf{u},z+1,y) + r(u_z,u_{z+1},z) \end{aligned} \tag{5.7}$$

Consequently, the cost function φ_α can also be decomposed for any x,y,z with $x\le z<y$:

$$\begin{aligned} \varphi_\alpha(\mathbf{u},x,y) &= (1-\alpha)R(\mathbf{u},x,y) + \alpha M(\mathbf{u},x,y) \\ &= (1-\alpha)\left[R(\mathbf{u},x,z) + R(\mathbf{u},z+1,y) + r(u_z,u_{z+1},z)\right] + \alpha\left[M(\mathbf{u},x,z) + M(\mathbf{u},z+1,y)\right] \\ &= \varphi_\alpha(\mathbf{u},x,z) + \varphi_\alpha(\mathbf{u},z+1,y) + (1-\alpha)r(u_z,u_{z+1},z) \end{aligned} \tag{5.8}$$

□

5.3.1 Forward Recursion Equations

The forward recursions construct an (α,p)-prefix optimal path $\mathbf{u}$ ending in sequence u_p based on an $(\alpha,p\text{-}1)$-prefix optimal paths $\mathbf{v}$ ending in some sequence v_{p-1}. In other words, the dynamic program only considers two columns in the alignment at a time and finds the transitions (u_{p-1}, u_p) at position p, such that path $\mathbf{u}$ is (α,p)-prefix optimal. It is quite easy to derive the forward recursion equations for computing the cost of a prefix optimal path:

$$\begin{aligned} f_{u_p,p} &= \min_{u_1,\ldots,u_{p-1}} \{\varphi_\alpha(\mathbf{u},1,p)\} = \min_{u_{p-1}} \left\{ \min_{u_1,\ldots,u_{p-2}} \{\varphi_\alpha(\mathbf{u},1,p)\} \right\} \\ &= \min_{u_{p-1}} \left\{ \min_{u_1,\ldots,u_{p-2}} \{\varphi_\alpha(\mathbf{u},1,p-1) + \varphi_\alpha(\mathbf{u},p,p) + (1-\alpha)r(u_{p-1},u_p,p-1)\} \right\} \\ &= \min_{u_{p-1}} \left\{ \min_{u_1,\ldots,u_{p-2}} \{\varphi_\alpha(\mathbf{u},1,p-1)\} + \varphi_\alpha(\mathbf{u},p,p) + (1-\alpha)r(u_{p-1},u_p,p-1) \right\} \\ &= \min_{u_{p-1}} \left\{ f_{u_{p-1},p-1} + \varphi_\alpha(\mathbf{u},p,p) + (1-\alpha)r(u_{p-1},u_p,p-1) \right\} \\ &= \min_{u_{p-1}} \left\{ f_{u_{p-1},p-1} + (1-\alpha)r(u_{p-1},u_p,p-1) \right\} + \alpha m(u_p,p) \end{aligned} \tag{5.9}$$

The recursion is terminated by the cost $f_{u_1,1} = \alpha m(u_1,1)$ of an $(\alpha,1)$-prefix optimal path ending in any sequence $u_1\in\{1,\ldots,M\}$. We can rewrite the recursion equations without referencing a path $\mathbf{u}$, as the equations only depend on the source and destination sequences j

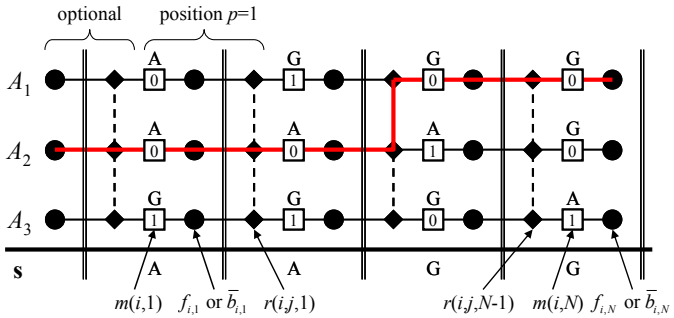


Figure 5.2: The structure of the dynamic program tableau is shown for the forward recursions. The circles represent the cost f_{ip} and $\bar{b}_{ip}$ in the tableau. The diamonds and the boxes specify the place at which recombination and mutation occur, respectively. Inside the boxes, the mutation cost relative to the putative recombinant **s** is given. The vertical double lines separate the stages of the dynamic program and also visualize that the actual sequence A_{ip} corresponds to the values f_{ip} and $\bar{b}_{ip}$. An α-optimal path **u**=(2,2,1,1) for α=1 is marked with a red line and explains the putative recombinant **s** by A_1 and A_2 using one recombination and no mutation event. The forward recursion initializes the leftmost circles with 0 and solves the dynamic program from left to right. The value for each circle is computed by considering all possible transitions from the circles one position to the left.

and i, respectively. For implementation, it is also more convenient to introduce a dummy state at position 0 with $f_{i,0}$=0 for all sequences i and define $r(i,j,0)$=0 for all sequences i and j, resulting in the following recursion equations

$$\begin{aligned} f_{ip} &= \min_j \left\{ f_{j,p-1} + (1-\alpha) r(j,i,p-1) \right\} + \alpha m(i,p) \\ f_{i,0} &= 0 \end{aligned} \quad . \tag{5.10}$$

The tableau shown in Figure 5.2 is a more intuitive representation of the recursion equations above. The dynamic program then computes the optimal values for f_{ip} from left to right in the tableau. The cost of an α-optimal path is the minimum of f_{iN} over i – the minimal cost of an (α,N)-prefix optimal path ending in any sequence i. Section 5.3.3 presents details on recovering the α-optimal paths from the tableau.

The forward recursions require $O(NM^2)$ time and $O(NM)$ space for the tableau. For the common case if $r(i,j,p)=\delta(i,j)r(p)$ and $r(p)\geq 0$ for all p the time complexity can be reduced to $O(NM)$. The idea is to precompute the minimum of $f_{j,p-1}$ regarding j once

$$MinCost(p) = \min_i \left\{ f_{ip} \right\} \tag{5.11}$$

and rewrite the forward recursion equations to handle the case $i=j$ in the minimization separately:

$$
\begin{aligned}
f_{ip} &= \min_j \left\{ f_{j,p-1} + (1-\alpha)\delta(i,j)r(p-1) \right\} + \alpha m(i,p) \\
&= \min \left\{ \min_{j \neq i} \left\{ f_{j,p-1} + (1-\alpha)\delta(i,j)r(p-1) \right\}, f_{i,p-1} + (1-\alpha)\delta(i,i)r(p-1) \right\} + \alpha m(i,p) \\
&= \min \left\{ \min_{j \neq i} \left\{ f_{j,p-1} \right\} + (1-\alpha)r(p-1), f_{i,p-1} \right\} + \alpha m(i,p) \\
&= \min \left\{ MinCost(p-1) + (1-\alpha)r(p-1), f_{i,p-1} \right\} + \alpha m(i,p)
\end{aligned}
\tag{5.12}
$$

In other words, either there is a transition without recombination from $f_{i,p-1}$ to f_{ip} or there is a transition with recombination to f_{ip} from one of the $f_{j,p-1}$ involving minimal cost. Both cases can also occur simultaneously.

5.3.2 Backward Recursion Equations

The backward recursion equations for recovering the cost of suffix optimal paths are derived analogously to the forward recursion equations for prefix optimal paths:

$$
\begin{aligned}
b_{ip} &= \min_{u_{p+1},\ldots,u_N} \left\{ \varphi_\alpha(\mathbf{u}, p, N) \right\} = \min_j \left\{ b_{j,p+1} + (1-\alpha)r(i,j,p) \right\} + \alpha m(i,p) \\
b_{iN} &= 0
\end{aligned}
\tag{5.13}
$$

The backward recursions do not correspond to the tableau shown in Figure 5.2, as b_{ip} measures the cost ending in sequence i including the mutation cost for position p. It is sometimes more intuitive to define backward recursions corresponding to the tableau in Figure 5.2, particularly for the forward-backward analysis presented in section 5.4.1:

$$
\begin{aligned}
\bar{b}_{ip} &= b_{ip} - \alpha m(i,p) \\
&= \min_j \left\{ b_{j,p+1} + (1-\alpha)r(i,j,p) \right\} + \alpha m(i,p) - \alpha m(i,p) \\
&= \min_j \left\{ \bar{b}_{j,p+1} + \alpha m(j,p+1) + (1-\alpha)r(i,j,p) \right\} \\
\bar{b}_{iN} &= 0
\end{aligned}
\tag{5.14}
$$

The same idea as for the forward recursion leads to a faster algorithm if $r(i,j,p)=\delta(i,j)r(p)$ and $r(p) \geq 0$ for all p. The time complexity for the backward recursions is $O(NM)$ if the minimum of b_{ip} over i is precomputed:

$$
b_{ip} = \min \left\{ \min_j \left\{ b_{j,p+1} \right\} + (1-\alpha)r(p), b_{i,p+1} \right\} + \alpha m(i,p) \tag{5.15}
$$

and substituting $b_{ip} = \bar{b}_{ip} + \alpha m(i,p)$ gives

$$
\bar{b}_{ip} = \min \left\{ \min_j \left\{ \bar{b}_{j,p+1} + \alpha m(j,p+1) \right\} + (1-\alpha)r(p), \bar{b}_{i,p+1} + \alpha m(i,p+1) \right\}. \tag{5.16}
$$

5.3.3 Recovering the Set of α-Optimal Paths

There is often more than one α-optimal path for the optimization problem, particularly if α is large and if there are many recombinations in the α-optimal paths. Recovering only a single α-optimal path is not acceptable in this case. The recombination breakpoint can vary greatly between different α-optimal paths, and a single α-optimal path does not suffice to describe the solution space. Unfortunately, the number of α-optimal paths grows very quickly with increasing α, so that it is not possible to recover all α-optimal paths in general. There are two approaches to this problem: we can summarize the set of α-optimal paths and only recover the positions of the recombination breakpoints.

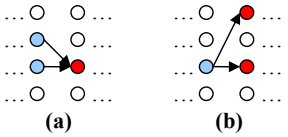


Figure 5.3: Two adjacent columns of the dynamic programming tableau are shown, where circles correspond to the values f_{ip} as in Figure 5.2. It is sufficient to focus on transitions between any two adjacent columns in the dynamic program, as the rest of the tableau is immaterial for finding the optimal transition between these columns. **(a)** Only transitions to the red circle are visualized. The recursion equations for the red circle are only satisfied for transitions coming from the blue circles. The paths passing through the blue circles from the left are independent from each other, unless they merge again as shown in (b). **(b)** Only transitions from the blue circle are visualized. The two paths running through the red circles from the right are independent from each other. However, both paths merge at the blue circle and are equivalent to the left of the blue circle. Both scenarios (a) and (b) can occur in the same column, though.

Alternatively, we can visualize the set of α-optimal paths intuitively if $r(i,j,p)=\delta(i,j)r(p)$. In any case, the forward recursions and the backward recursions can be used to recover the set of α-optimal paths by backtracing. The approach is similar for both, so that only the forward recursions are presented. The backtracing step described in the following can recover all (α,p)-prefix optimal paths for all p. The α-optimal paths are then recovered by backtracing the (α,N)-prefix optimal paths with minimum cost.

5.3.3.1 Recovering all α-Optimal Paths and Their Breakpoint Positions

First, consider the problem of reconstructing all α-optimal paths. The structure of the dynamic program enables us to focus on two adjacent columns of the alignment at a time (see Figure 5.3), where each column in the alignment also corresponds to a column in the dynamic program. The tableau f_{ip} only stores the cost of the (α,p)-prefix optimal paths ending in sequence i and for all p, but it is easy to reconstruct the transitions on these paths. The transitions T_p occurring on any (α,p)-prefix optimal paths from any sequence j at position p-1 to any sequence i at position p then directly follows from the recursion equations (5.10):

$$T_p = \left\{(j,i) \mid f_{ip} = f_{j,p-1} + (1-\alpha)r(j,i,p-1) + \alpha m(i,p)\right\}. \tag{5.17}$$

It is convenient to define $T_p(i)=\{(j,i')\in T_p|i'=i\}$ as the set of transitions ending in sequence i. It is easy to see that $|T_p(i)|\geq 1$ for all i as there is at least one (α,p)-prefix optimal path ending in every sequence i. A simple recursive algorithm suffices to recover all (α,p)-prefix optimal paths for any p ending in sequence i:

recover_paths(p,i)

initialize U={}

if (p>1) set $U = \{recover\_paths(p-1, j) \mid (j,i) \in T_p(i)\}$

else set U={**u**} where **u** is an arbitrary path

for every **u**$\in U$ set u_p=i

return U

Now define *MinCostSeq*(p) as the set of (indices of) sequences, for which f_{ip} is minimal for position p:

$$MinCostSeq(p) = \arg\min_i \{f_{ip}\} = \{i \mid f_{ip} = MinCost(p)\} \tag{5.18}$$

The set of α-optimal paths is then equivalent to the set of (α,N)-prefix optimal paths ending in any sequence $i \in MinCostSeq(N)$ as these prefix optimal paths involve the minimum cost overall. As described before, the number of paths grows too quickly and the *recover_paths* algorithm is too slow in practice. Fortunately, it is possible to recover the set of all recombination breakpoints for all α-optimal paths in O(NM^2) time. Starting with I=*MinCostSeq*(N), I is updated to represent the set of sequences on α-optimal paths at position p-1 by tracing back the transitions in T_p:

recover_breakpoints:

initialize I=*MinCostSeq*(N) and set *breakpoint*(p)=false for all p

for every p=N,…,2 do

set *breakpoint*(p)=true if there exists $i \in I$ and a transition $(j,i) \in T_p(i)$ with $j \neq i$

initialize $I \leftarrow \{j \mid (j,i) \in T_p(i)$ and $i \in I\}$ to the set of sequences j that are reachable from i

return *breakpoint*

5.3.3.2 The α-Optimal Paths for Simple Recombination Costs

Suppose that $r(i,j,p)=\delta(i,j)r(p)$ and that we are interested in the transitions on (α,p)-optimal paths. As the recursion equations can be rewritten as in (5.12), the set of transitions T_p on (α,p)-optimal paths satisfies

$$\begin{aligned} T_p = &\{(j,i) \mid j \in MinCostSeq(p-1) \wedge f_{ip} = MinCost(p-1) + (1-\alpha)r(p-1) + \alpha m(i,p)\} \\ &\cup \{(i,i) \mid f_{ip} = f_{i,p-1} + \alpha m(i,p)\} \end{aligned} . \tag{5.19}$$

T_p is much more constrained than before. First, focus on the transitions $(j,i) \in T_p(i)$ that involve a recombination and therefore satisfy $j \neq i$. It directly follows from (5.12) that a transition $(j,i) \in T_p(i)$ with $j \neq i$ must satisfy $j \in MinCostSeq(p\text{-}1)$, as only in this case f_{ip}=*MinCost*(p-1)+(1-α)r(p-1). All transitions (j',i) with $j' \in MinCostSeq(p\text{-}1)$ are also cost minimal and satisfy $(j',i) \in T_p(i)$. Any other transition (j,i) with $j \neq i$ and $j \notin MinCostSeq(p\text{-}1)$ incurs a higher cost and is not contained in the set $T_p(i)$. The essential insight is that two

sequences i and i' allowing for a transition with recombination at position p both satisfy $(j,i)\in T_p(i)$ and $(j,i')\in T_p(i')$ for the same sequences $j\in MinCostSeq(p\text{-}1)$. Transitions (j,i) without recombination satisfy $j=i$ and are independent of the discussion above. These insights help to formalize the structure inherent to T_p. Define

$$\begin{aligned} RecSeq(p) &= \{i \mid (j,i) \in T_p \wedge i \neq j\} \\ NoRecSeq(p) &= \{i \mid (i,i) \in T_p\} \end{aligned} \tag{5.20}$$

where $RecSeq(p)$ corresponds to sequences i such that there is a transition with recombination ending in sequence i, and $NoRecSeq(p)$ corresponds to sequences i such that a transition with recombination is not necessary. Some sequences i may satisfy both, $i\in RecSeq(p)$ and $i\in NoRecSeq(p)$.

As shown above, a sequence $i\in RecSeq(p)$ allows for transitions from any sequence $j\in MinCostSeq(p\text{-}1)$. All other sequences $j\notin MinCostSeq(p\text{-}1)$ with $j\neq i$ do not lead to an optimal transition $(j,i)\in T_p(i)$. Transitions $(i,i)\in T_p(i)$ occur independently and for all $i\in NoRecSeq(p)$. Hence, the set of transitions T_p can also be represented as follows:

$$T_p = \{(j,i) \mid j \in MinCostSeq(p-1) \wedge i \in RecSeq(p)\} \cup \{(i,i) \mid i \in NoRecSeq(p)\}. \tag{5.21}$$

Consequently, it is sufficient to identify the set of source sequences $MinCostSeq(p\text{-}1)$, the set of destination sequences $RecSeq(p)$ and the sequences in $NoRecSeq(p)$ to recover all transitions in T_p (see Figure 5.4(a)). Furthermore, several simple arguments lead to the following relations:

(a) $RecSeq(p)\cup NoRecSeq(p)=\{1,\ldots,M\}$

This is true, because there is at least one transition ending in an arbitrary sequence $i\in\{1,\ldots,M\}$.

(b) $RecSeq(p)\cap MinCostSeq(p\text{-}1)=\varnothing$

Suppose there is a sequence i with $i\in MinCostSeq(p\text{-}1)$ and $i\in RecSeq(p)$. It then follows from (5.19) that the transition (i,i) incurs less cost than any other transition (j,i) for $j\neq i$. Hence, $(j,i)\notin T_p$ and i cannot be an element of $RecSeq(p)$.

(c) $NoRecSeq(p)\supseteq MinCostSeq(p\text{-}1)$

This directly follows from (5.19): if $i\in MinCostSeq(p\text{-}1)$, $f_{i,p\text{-}1}$ is minimal and the transition (i,i) is cost optimal. Every other transition (j,i) with $j\neq i$ incurs a higher cost, as it introduces a recombination. Hence, $i\in NoRecSeq(p\text{-}1)$ for every $i\in MinCostSeq(p\text{-}1)$.

(d) $\{j|(j,i)\in T_p\}=NoRecSeq(p)$

In other words, all transitions from p-1 to p begin at a sequence in $NoRecSeq(p)$. The reasoning is simple: from (5.21) it follows that $\{j|(j,i)\in T_p\}=NoRecSeq(p)\cup MinCostSeq(p\text{-}1)$, and the proposition follows from (c), because $NoRecSeq(p)\supseteq MinCostSeq(p\text{-}1)$.

We are usually only interested in the set of transitions on all α-optimal paths and therefore we only need to identify and track a subset of $RecSeq(p)$ and $NoRecSeq(p)$ at each

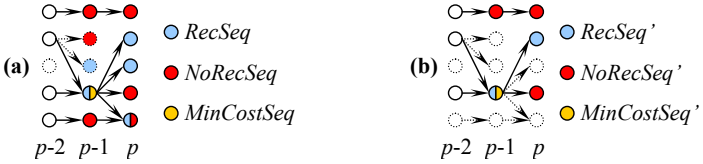


Figure 5.4: The same example for the structure of the transitions in T_p and T_{p-1} is shown in all plots, but the coloring scheme differs slightly between plots. The arrows represent all transitions in T_p and T_{p-1}. Dotted arrows and circles represent transitions and states that are not of interest for the analysis given in a plot. **(a)** Dotted arrows and circles do not lie on an (α,p)-prefix optimal path and therefore are not reachable if we trace back all circles from the rightmost column to the leftmost column following arrows in the opposite direction. The coloring of the circles visualizes the association to the different sets introduced in the text and some circles are split to indicate that they belong to two sets. As the transitions T_{p-2} are not given, we cannot color the circles in the leftmost column accordingly. It is sufficient to identify sequences in *RecSeq*(p), *NoRecSeq*(p) and *MinCostSeq*(p-1) to recover the transitions in T_p. For example, there are always transitions from all sequences in *MinCostSeq* to all sequences in *RecSeq*. **(b)** In comparison to the previous plot, dotted arrows and circles indicate that they do not lie on an α-optimal path and are therefore also uncolored. In this example, the third and the fifth sequence counting from top at position p do not lie on an α-optimal path and trigger several other sequences to become dotted for position p-1.

position p. To be more specific, we can define the set of sequences on the α-optimal paths at each position recursively:

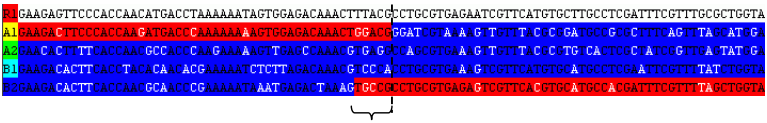
$$\begin{aligned} OptPathSeq(N) &= MinCostSeq(N) \\ OptPathSeq(p-1) &= \left\{ j \mid (j,i) \in T_p \wedge i \in OptPathSeq(p) \right\} \end{aligned} \tag{5.22}$$

and define the following restricted sets

$$\begin{aligned} MinCostSeq'(p) &= MinCostSeq(p) \cap OptPathSeq(p) \\ RecSeq'(p) &= RecSeq(p) \cap OptPathSeq(p) \\ NoRecSeq'(p) &= NoRecSeq(p) \cap OptPathSeq(p) \end{aligned} \quad . \tag{5.23}$$

Figure 5.4 (b) visualizes only these sets and allows to focus exclusively on the α-optimal paths. It is possible to implement this coloring scheme in Recco, but it is impractical to use two colors for each nucleotide. Hence, Recco implements a much simpler coloring scheme and only uses a single color for the nucleotides on all α-optimal paths. In general, it is not possible to reconstruct all transitions along the α-optimal paths from such a representation. But recombination breakpoints are often clearly visible in practice as shown in Figure 5.5.

The special structure of T_p also allows for recovering the set of recombination breakpoints on α-optimal paths in O(NM) time. As before, the algorithm traces the set of sequences I on α-optimal paths:



predicted breakpoint locations

Figure 5.5: The α-optimal paths are shown for query sequence R1 using α=0.2 and Hamming distances for mutation and recombination cost. The background color of the nucleotides visualizes the result of the optimization problem: white represents the query sequence, red denotes nucleotides on an α-optimal path, and any other nucleotide is shown with a blue background. A white foreground color visualizes mutations regarding the query sequence. The true recombination breakpoint occurs in the middle of the alignment and is shown by the black, dotted line.

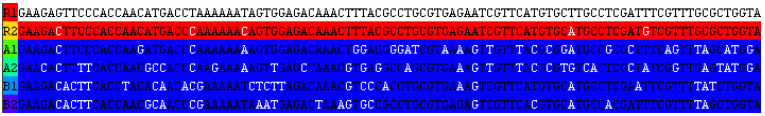


Figure 5.6: The α-optimal path for the alignment including sequence R2 uses R2 for explaining R1.

recover_breakpoints:

initialize I=*MinCostSeq*(N)

initialize *breakpoint*(p)=false for all p

for every p=N,…,2 do

set *breakpoint*(p)=true if there exists $i \in I$ with f_{ip}=*MinCost*(p-1)+(1-α)r(p-1)+αm(i,p)

if (*breakpoint*(p)=true) $I \leftarrow$(*NoRecSeq*(p)$\cap I$)$\cup$*MinCostSeq*(p-1)

else $I \leftarrow$*NoRecSeq*(p)$\cap I$

return *breakpoint*

5.3.4 Examples of α-Optimal Paths

Figure 5.5 shows the α-optimal paths for the example alignment described in section 5.1. The optimization run used Hamming distances for mutation and recombination cost and set α to 0.2, such that each recombination costs as much as (1-0.2)/0.2=4 mutations. In other words, a recombination only pays off if it avoids four or more mutations. All α-optimal paths use sequence A1 for the first part of the alignment and sequence B2 for the rest of the alignment. The paths only differ regarding the location of the recombination breakpoint. The breakpoints are predicted at several, but not at all positions within

the marked range. For example, a recombination breakpoint from sequence A1 to B2 between the two G's in A1 with a white foreground leads to a higher mutation cost and is not optimal. However, this effect is not explicitly shown in Figure 5.6.
The α-optimal paths closely resemble the true ancestry of R1, but wrongly predict B2 as the sequence ancestral to R1 in the right part of the alignment. Now consider the full alignment including R2. What is the expected α-optimal path if R1 is the query sequence? Apparently, R2 is very similar to R1 for the whole alignment, and therefore it is more parsimonious to explain R1 without recombination. The α-optimal path simply covers R2 and does not require many mutations (see Figure 5.6).

5.3.5 Speedup by Preprocessing the Alignment

If the mutation cost is the same for all sequences in a column, we can actually remove the position from the input. The relevant observation is given in the following Lemma, where $\Delta(\mathbf{u})$ refers to the set of recombination breakpoints of path $\mathbf{u}$ as defined before.

Lemma: Assume that $r(i,j,p)=\delta(i,j)r(p)$ and suppose that $m(i,p)=const$ for all i and all $p\in[a,b]$. Define the positions of minimum recombination cost in the interval $[a,b]$ as $X = \arg\min_{p\in[a,b]}\{r(p)\}$. Given an α-optimal path $\mathbf{u}$, it follows that

- **(a)** $|\Delta(\mathbf{u})\cap[a,b]|\leq 1$ (there is at most one recombination in the interval $[a,b]$), and
- **(b)** if $|\Delta(\mathbf{u})\cap[a,b]|=1$ and $x\in\Delta(\mathbf{u})\cap[a,b]$, then a path $\mathbf{v}$ that moves the breakpoint to any position $y\in X$ is also α-optimal.

Proof:
(a) Suppose that $|\Delta(\mathbf{u})\cap[a,b]|>1$. Hence, there exist two breakpoints $x,y\in[a,b]$ with $x<y$. But the path that only introduces a recombination breakpoint at position y involves less cost. To be more specific, define path $\mathbf{v}$ with $v_p=u_p$ for all $p\in[1,x]\cup[y+1,N]$ and $v_p=u_x$ for all $p\in[x+1,\ y]$. From the separability of the cost function (see (5.8)) and as $\varphi_\alpha(\mathbf{u},1,x)=\varphi_\alpha(\mathbf{v},1,x)$ and $\varphi_\alpha(\mathbf{u},y+1,N)=\varphi_\alpha(\mathbf{v},y+1,N)$ it follows that

$$\begin{aligned}\varphi_\alpha(\mathbf{u},1,N)-\varphi_\alpha(\mathbf{v},1,N) &= \varphi_\alpha(\mathbf{u},x+1,y)+\delta(u_x,u_{x+1})r(x)+\delta(u_y,u_{y+1})r(y)\\ &\quad -\varphi_\alpha(\mathbf{v},x+1,y)-\delta(v_x,v_{x+1})r(x)-\delta(v_y,v_{y+1})r(y)\\ &= \left[\delta(u_x,u_{x+1})-\delta(v_x,v_{x+1})\right]r(x)+\left[\delta(u_y,u_{y+1})-\delta(v_y,v_{y+1})\right]r(y)\end{aligned}$$

and resolving $\delta(v_x,v_{x+1})=0$ yields

$$\varphi_\alpha(\mathbf{u},1,N)-\varphi_\alpha(\mathbf{v},1,N)=r(x)+r(y)-r(y)>0\,.$$

This contradicts the assumption that path $\mathbf{u}$ is α-optimal and it follows that $|\Delta(\mathbf{u})\cap[a,b]|\leq 1$.

(b) Now suppose that there is exactly one recombination breakpoint $x \in \Delta(\mathbf{u}) \cap [a,b]$ and let $y \in X$. If $x<y$, we can define a path $\mathbf{v}$ as above: $v_p=u_p$ for all $p \in [1,x] \cup [y+1,N]$ and $v_p=u_x$ for all $p \in [x+1, y]$. As $y \in X$, it follows that the cost difference is

$$\begin{aligned}\varphi_\alpha(\mathbf{u},1,N) - \varphi_\alpha(\mathbf{v},1,N) &= \left[\delta(u_x,u_{x+1}) - \delta(v_x,v_{x+1})\right] r(x) + \left[\delta(u_y,u_{y+1}) - \delta(v_y,v_{y+1})\right] r(y) \\ &= r(x) - r(y) \geq 0\end{aligned}$$

and $\mathbf{v}$ is also α-optimal. The case for $y<x$ is analogous.

□

In other words, if there is a region of columns $p \in [a,b]$ with $m(i,p)=const$ for all i and all $p \in [a,b]$, the dynamic program does not have to process the columns in $[a,b]$. Instead, we define an alternative recombination cost r' that only differs from r at position $b+1$: $r(b+1)=\min\{r(p) | p \in [a,b]\}$. We can then solve an alternative dynamic program using r' as a recombination cost and skip columns a to b. In other words, $f_{i,b+1}$ depends on $f_{j,a-1}$ and not on f_{jb}. If the alternative dynamic program predicts a recombination breakpoints exactly before position $b+1$, then the original dynamic program introduces recombination breakpoints for all positions $\arg\min_{p \in [a,b]} \{r(p)\}$.

5.4 Sensitivity Analysis

We are often not only interested in the solution of an optimization problem, but also in a sensitivity analysis of the solution. There are two fundamentally different approaches for a sensitivity analysis in our case. The first approach is subsequently called sensitivity analysis of the α-optimal paths and studies the cost of paths that are cost optimal for a restricted configuration space. The restricted configuration space we consider does not allow the paths to include a certain nucleotide in the alignment and can tell us whether the nucleotide is critical for attaining the low cost of the α-optimal paths. As an example, assume that the α-optimal paths incur a much lower cost than the cost optimal paths that avoid a certain nucleotide. In this case, we know that the nucleotide is very important for the low cost of the α-optimal paths and represents a robust part of the α-optimal paths.

The second approach for a sensitivity analysis is subsequently called parametric analysis. A parametric analysis studies how the solution depends on the input parameters of the optimization problem, such as α, $m(i,p)$ and $r(i,j,p)$ in our case. More exactly, a parametric analysis partitions the space of parameter settings into regions that yield the same solution. Hence, a parametric analysis helps to assess the impact of uncertainty of parameters, e.g. due to measurement or estimation errors, on the structure of the solution. It is particularly rewarding to study the parameter α for the given optimization problem, as it allows for measuring the strength of the recombination signal in the query sequence.

Apparently, both approaches address different questions regarding the stability of the α-optimal paths. Both approaches have also been applied separately to the problem of sequence alignment (Mevissen and Vingron 1996; Zimmer and Lengauer 1997).

5.4.1 Sensitivity Analysis of the *α*-Optimal Paths

In the following, I transfer an idea by Mevissen and Vingron (Mevissen and Vingron 1996) from traditional sequence alignment to the optimization problem introduced in the previous section. Mevissen and Vingron describe a method for assessing the robustness of a pairwise sequence alignment. In a first step, they compute the minimum cost c_{ij} of a pairwise alignment matching position i of the first sequence with position j of the second sequence. Computation of c_{ij} is very efficient and only requires the forward and backward tableaus as input. The second step computes the minimum cost d_{ij} of a pairwise alignment that does not match position i with position j. The robustness value of pairing position i with position j is then $r_{ij}=c_{ij}-d_{ij}$ and is positive for pairings on the optimal path. A high robustness value also indicates that the pairing is important for attaining a low overall cost.

As the optimization problem of Recco determines a path $\mathbf{u}$ through the alignment, robustness is defined regarding the choice of sequence u_p for position p. The first step is therefore to compute the minimum cost c_{ip} of a path running through nucleotide A_{ip} in sequence i at position p. The minimal cost of such a path $\mathbf{u}$ with $u_p=i$ can be computed from the forward and the backward tableau:

$$\begin{aligned} c_{ip} &= \min_{\mathbf{u},u_p=i}\{\varphi_\alpha(\mathbf{u},1,N)\} \\ &= \min_{\mathbf{u},u_p=i}\{\varphi_\alpha(\mathbf{u},1,p)+\varphi_\alpha(\mathbf{u},p,N)-\alpha m(u_p,p)\} \\ &= f_{ip}+b_{ip}-\alpha m(i,p) \\ &= f_{ip}+\bar{b}_{ip} \end{aligned} \tag{5.24}$$

The minimum cost d_{ip} of a path avoiding sequence i at position p is obtained analogously

$$\begin{aligned} d_{ip} &= \min_{\mathbf{u},u_p\neq i}\{\varphi_\alpha(\mathbf{u},1,N)\} \\ &= \min_{\mathbf{u},u_p\neq i}\{\varphi_\alpha(\mathbf{u},1,p)+\varphi_\alpha(\mathbf{u},p,N)-\alpha m(u_p,p)\} \\ &= \min_{j\neq i}\{f_{jp}+b_{jp}-\alpha m(j,p)\} \\ &= \min_{j\neq i}\{f_{jp}+\bar{b}_{jp}\} = \min_{j\neq i}\{c_{jp}\} \end{aligned} \tag{5.25}$$

It is more intuitive to use the values $\bar{b}_{ip}$ and f_{ip} corresponding to the same tableau for these equations, as these values refer to the same circle in Figure 5.2 and therefore their sum corresponds to a complete path from left to right running through a specific circle. In analogy to (Mevissen and Vingron 1996), the robustness measure is then defined as $r_{ip}=c_{ip}-d_{ip}$ for every i and p. In words, r_{ip} is the sacrifice in cost that is incurred by forcing a solution to avoid position p in sequence i in our alignment. If r_{ip} is small then the choice of position p in sequence i does not buy us much in terms of alignment cost. Larger values of r_{ip} indicate more robust sequence positions. The values c_{ip} and r_{ip} can be computed in time O(MN) for all combinations of p and i if the forward and backward tableau f_{ip} and b_{ip} are given.

Both values, c_{ip} and r_{ip}, are useful in our scenario (see Figure 5.7). c_{ip} is particularly attractive for visualizing the optimal paths of the dynamic program. It is easy to compute

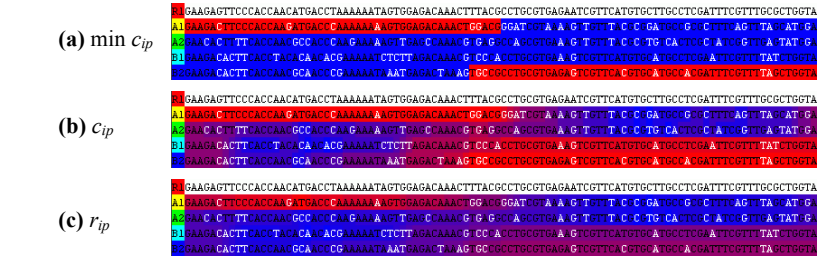


Figure 5.7: The variability of the α-optimal paths for explaining sequence R1 is visualized by different approaches. **(a)** Nucleotides corresponding to the minimal c_{ip} are shown with a red background as in Figure 5.5. **(b)** A red (blue) background indicates a low (high) value of c_{ip}. The analysis suggests that it is possible to construct a path using B1 instead of B2 for the right part of the alignment without affecting the cost greatly. This also matches the expected result. **(c)** Red and blue backgrounds indicate high and low robustness values r_{ip}, respectively. The α-optimal paths are less reliable at the predicted recombination breakpoint and in the right part of the alignment.

the representation of the α-optimal paths shown in Figure 5.5, as all nucleotides A_{ip} that are part of an α-optimal path have the same, minimal c_{ip}. Alternatively, we can visualize the c_{ip} by coloring the background of the alignment A_{ip} according to c_{ip}. For positions at which the solution is uncertain or close to a recombination breakpoint, low values of c_{ip} occur for several sequences to indicate the ambiguity. On the other hand, if we ask how robust the choice of a specific nucleotide A_{ip} is, we can rely on the values of r_{ip}. Regions that are uncertain are characterized by a lower robustness of the optimal solution.

5.4.2 Parametric Analysis Regarding α

The α-optimal paths, or the solutions of the optimization problem, depend on the parameters α, $m(i,p)$ and $r(i,j,p)$. The parameter α controls the amount of recombination in α-optimal paths and is the only parameter studied in the following for several reasons. First, setting $m(i,p)$ and $r(i,j,p)$ to Hamming distances usually gives good results in practice and also allows for a simple interpretation of the parameter α. In comparison to $m(i,p)$ and $r(i,j,p)$ it is also much more difficult to set the parameter α appropriately, as α depends on the relative sizes of recombination rate and mutation rate in the organism or virus under study. Second, α has the most significant impact on the α-optimal paths, as it determines the amount of recombination in the paths. The parametric analysis of α also has a clear interpretation regarding the recombination signal in the alignment and allows for extracting statistics for recombination analysis automatically. Third, an analysis of two or more parameters leads to a multi-dimensional output that is much harder to comprehend or to process automatically. Finally, analyzing more than one parameter requires

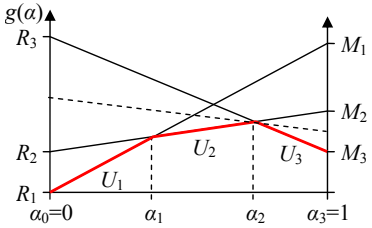


Figure 5.8: The parametric cost curve g is shown in red. The y-axis and x-axis represent the cost of $g(\alpha)$ and α, respectively. The recombination cost R_i and the mutation cost M_i of a path $\mathbf{u} \in U_i$ correspond to the abscissa of the line segment of U_i at α=0 and α=1, respectively, and both points also completely define the line. The dotted line segment corresponds to a degenerate α-optimal path for $\alpha=\alpha_2$ and is not studied in the parametric analysis.

complex algorithms and may require a careful assessment of the numerical accuracy (Zimmer and Lengauer 1997).

5.4.2.1 The Structure of the Cost Curve

A parametric analysis regarding α investigates how $\varphi_\alpha(\mathbf{u},1,N)$ changes as α is varied, both for a fixed solution $\mathbf{u}$ and for $\mathbf{u}$ subject to optimization. If $\mathbf{u}$ is fixed, $\varphi_\alpha(\mathbf{u},1,N)$ is linear in α and has the derivative

$$\frac{\partial \varphi_\alpha(\mathbf{u},1,N)}{\partial \alpha} = -R(\mathbf{u},1,N) + M(\mathbf{u},1,N) \tag{5.26}$$

where $R(\mathbf{u},1,N)$ and $M(\mathbf{u},1,N)$ are the recombination and mutation cost of path $\mathbf{u}$ not weighted by (1-α) or α. Now, define the minimum cost of $\varphi_\alpha(\mathbf{u},1,N)$ if $\mathbf{u}$ is subject to optimization as

$$g(\alpha) = \min_{\mathbf{u}} \{\varphi_\alpha(\mathbf{u},1,N)\} \tag{5.27}$$

As φ_α is linear in α for every path $\mathbf{u}$, $g(\alpha)$ is a piecewise-linear concave function in α (see Figure 5.8). Each linear piece corresponds to a fixed set of minimum cost solutions. Clearly, the function value $g(\alpha)$ and the corresponding α-optimal paths can be computed for every α with the dynamic program presented before.

Now define K as the number of segments of $g(\alpha)$ and define $\alpha_0<\alpha_1<\ldots<\alpha_K$ as the segment boundaries where α_0=0 and α_K=1. The k-th segment covers $[\alpha_{k-1}, \alpha_k]$. Let U_k be the set of α-optimal paths for the k-th segment, such that $\mathbf{u} \in U_k$ is α-optimal for all $\alpha \in [\alpha_{k-1}, \alpha_k]$. In addition to the α-optimal paths in U_k, there may exist several degenerate α-optimal paths. A degenerate α-optimal path $\mathbf{u}$ is only α-optimal for one specific $\alpha \in \{\alpha_0,\ldots,\alpha_K\}$. Degenerate α-optimal paths are not of interest here, but complicate the algorithm for the parametric analysis. All paths $\mathbf{u} \in U_k$ incur the same amount of mutation and recombination cost, as they correspond to the same segment of $g(\alpha)$ and have the same derivative. Hence, we

can define the recombination cost as $R_k=R(\mathbf{u},1,N)$ and the mutation cost as $M_k=M(\mathbf{u},1,N)$ for every k, where $\mathbf{u}$ is an arbitrary element in U_k. With increasing α and increasing k, the α-optimal paths $\mathbf{u}$ contain less mutation cost and more recombination cost and it follows that $0=R_1<\ldots<R_K$ and $M_1>\ldots>M_K$. The paths U_k are therefore ordered by increasing recombination cost and decreasing mutation cost.

It is interesting to understand the α-optimal paths for specific values of α. Clearly, one unit of recombination cost $R(\cdot)$ effectively contributes as much to the overall cost of the path as $(1-\alpha)/\alpha$ units of mutation cost $M(\cdot)$ for all α, because $\varphi_\alpha(\cdot)=(1-\alpha)R(\cdot)+\alpha M(\cdot)$. The α-optimal paths for $\alpha=0$ only use mutations for explaining the query sequence. Consequently, the line corresponding to U_1 always intersects the origin and $R_1=0$. However, there are several degenerate solutions for $\alpha=0$, as any path $\mathbf{u}$ without recombination incurs the same cost $\varphi_0(\mathbf{u},1,N)=0$. We can avoid this complication by using a small $\alpha>0$, for example $\alpha=\alpha_1/2$. Clearly, an $\alpha_1/2$-optimal path $\mathbf{u}=(i,\ldots,i)$ then simply selects a sequence i that is closest to the query sequence in terms of mutation cost only. The mutation cost of $\mathbf{u}$ is therefore

$$M_1 = \min_i \left\{ \sum_{p=1}^{N} m(i,p) \right\} \tag{5.28}$$

and the line corresponding to U_1 is defined as αM_1.

The structure of α-optimal paths and the cost curve g is also restricted if $\alpha=1$. To avoid the problem of degenerate solutions, assume that α is close enough to 1, for example $\alpha=(\alpha_{K-1}+1)/2$. An α-optimal path $\mathbf{u}$ then only minimizes the mutation cost

$$M_K = \min_{\mathbf{u}} \left\{ \sum_{p=1}^{N} m(u_p,p) \right\} = \sum_{p=1}^{N} \min_i \left\{ m(i,p) \right\}. \tag{5.29}$$

In other words, M_K sums the minimum mutation cost for each column. Interestingly, M_K and M_1 do not change if we use a column permutation of the original alignment as input.

Now assume that we use Hamming distances for mutation and recombination cost. If $\alpha>2/3$, it follows that $\alpha/(1-\alpha)>2$ and therefore each mutation costs more than two recombinations. It is never optimal to introduce a mutation for $\alpha>2/3$ if there is a matching nucleotide in the same column for some other sequence. Consequently, $\alpha_{K-1}\leq 2/3$ and there is only one line segment in the interval $[2/3,1]$. M_K counts the number of columns in which the query sequence has a unique nucleotide compared to all other sequences. In practice, the α-optimal paths in U_K are usually not relevant because of their strong preference for recombination.

We can extract even more information from the cost curve if we compare the α-optimal paths in U_k with U_{k+1}. Paths in U_{k+1} reduce the mutation cost by $M_k-M_{k+1}>0$ and increase the recombination cost by $R_{k+1}-R_k>0$ when compared to paths in U_k. In other words, each additional unit of recombination cost reduces the mutation cost by

$$Savings_k = \frac{M_k - M_{k+1}}{R_{k+1} - R_k} \tag{5.30}$$

and a high value of $Savings_k$ indicates that recombination is advantageous. If Hamming distances are used for mutation and recombination cost, $Savings_k$ simply measures how many mutations are eliminated for every additional recombination event in the path. For example, if R_2-R_1=2 and M_1-M_2=11, then there are two recombination events that each eliminate $Savings_1$=5.5 mutations. $Savings_k$ is also related to the cutpoint α_k of the lines corresponding to U_k and U_{k+1}. The cutpoint α_k satisfies

$$(1-\alpha_k)R_k + \alpha_k M_k = (1-\alpha_k)R_{k+1} + \alpha_k M_{k+1}. \tag{5.31}$$

Solving for α_k yields

$$\alpha_k = \frac{-R_k + R_{k+1}}{-R_k + R_{k+1} + M_k - M_{k+1}} = \frac{1}{Savings_k + 1} \tag{5.32}$$

or equally $Savings_k$=(1-α_k)/α_k. It follows that $Savings_1$>…>$Savings_{K-1}$, as the α_k's are ordered increasingly.

$Savings_1$ is the maximum of $Savings_k$ and is particularly interesting as it quantifies whether a query sequence prefers a path with recombination. It is also possible to bound $Savings_1$ from above:

$$Savings_1 = \frac{M_1 - M_2}{R_2 - R_1} = \frac{M_1 - M_2}{R_2 - 0} \le \frac{M_1 - M_K}{R_2}. \tag{5.33}$$

For the common case that Hamming distances are used for recombination, the bound simplifies to

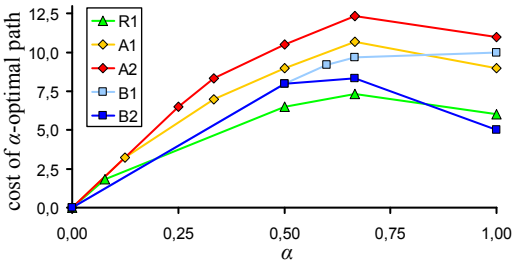
$$Savings_1 \le \frac{M_1 - M_K}{R_2} \le M_1 - M_K =: \sigma \tag{5.34}$$

as $R_2 \ge 1$ in this case. The bound σ is usually weak in practice, as it is rare that a single recombination (R_2=1) eliminates all mutations (K=2). A simple prediction is still possible: $Savings_1$ cannot be large if M_1 or σ is small – for example because there is a sequence similar to the query sequence in the alignment. A particularly attractive feature of the bound σ is that it is invariant with respect to column permutations, as M_1 and M_K do not change if the columns of the alignment are permuted.

We can also interpret α_k just as we did with $Savings_k$ and draw conclusions about the strength of the recombination signal from the α_k's alone, as there is a functional dependency between α_k and $Savings_k$. If the transition from U_k to U_{k+1} happens at a small value α_k, recombination events are introduced even though they are penalized strongly when compared to mutations. Each additional unit of recombination cost in U_{k+1} as compared to U_k decreases the mutation cost by (1-α_k)/α_k=$Savings_k$ units, as all paths in U_k and U_{k+1} are α_k-optimal. Hence, we know that there is no path increasing the recombination cost of any α-optimal path by one unit and reducing the mutation cost by more than α_1/(1-α_1) units, as the original α-optimal path would otherwise not be α-optimal.

5.4.2.2 Recovering the Parametric Cost Curve

We can use the algorithm of Eisner and Severance (Eisner and Severance 1976) to compute the parametric cost curve g efficiently, as the parametric analysis only involves a single parameter. The algorithm requires a method for recovering the cost $\varphi_\alpha(\mathbf{u},1,N)$ and



	$Savings_1$	σ	M_1
R1	12	18	24
A1	7	17	26
A2	3	15	26
B1	1	6	16
B2	1	11	16

Figure 5.9: **(left)** The parametric cost curves for each sequence in the example alignment are shown to the left. As α changes from 0 to 1, the cost for mutation increases and the cost for recombination decreases. The cost curve of R1 shows the highest benefit from recombination. A recombination is already introduced into the α-optimal paths for $\alpha_1 \approx 0.077$ and reduces the number of mutations by $Savings_1=12$ compared with the α-optimal path without recombination. **(right)** The table summarizes the observed $Savings_1$ values and also shows the bound σ. The $Savings_1$ value for R1 is particularly high when compared to σ. A1 also shows some preference for recombination, but the other sequences do not. The preference of A1 is also expected from the ARG (see section 5.5).

the derivative of $\varphi_\alpha(\mathbf{u},1,N)$ for an α-optimal path $\mathbf{u}$, as this defines the line corresponding to $\mathbf{u}$ (see (5.26)). The algorithm correctly handles degenerate solutions, such that it is sufficient to compute $R(\mathbf{u},1,N)$ and $M(\mathbf{u},1,N)$ of an arbitrary α-optimal path $\mathbf{u}$. Both values can therefore be computed easily by a modified backtrace algorithm.

The principle of the algorithm of Eisner and Severance is easily described by a recursion. However, the original algorithm is iterative. It starts with an interval $[x,y]$, an x-optimal path $\mathbf{u}_x$ and a y-optimal path $\mathbf{u}_y$. If both paths $\mathbf{u}_x$ and $\mathbf{u}_y$ yield the same line in the parametric cost curve, the concave function g is equal to this line for the full interval $[x,y]$ and the recursion terminates. Otherwise, compute the value z where the lines corresponding to $\mathbf{u}_x$ and $\mathbf{u}_y$ intersect and recover a z-optimal path $\mathbf{u}_z$. As the lines for $\mathbf{u}_x$ and $\mathbf{u}_y$ intersect at z, we know that $\varphi_z(\mathbf{u}_x,1,N)=\varphi_z(\mathbf{u}_y,1,N)$. The lines corresponding to $\mathbf{u}_x$ and $\mathbf{u}_y$ fully define g in the interval $[x,y]$ if $\varphi_z(\mathbf{u}_z,1,N)=\varphi_z(\mathbf{u}_x,1,N)=\varphi_z(\mathbf{u}_y,1,N)$. Otherwise $\varphi_z(\mathbf{u}_z,1,N)<\varphi_z(\mathbf{u}_x,1,N)$ and $\varphi_z(\mathbf{u}_z,1,N)<\varphi_z(\mathbf{u}_y,1,N)$ and the algorithm recursively processes the intervals $[x,z]$ and $[y,z]$.

The speed of this algorithm is dominated by the number of calls to the dynamic program for recovering an α-optimal path. The algorithm requires one call for every cutpoint z between two adjacent lines, as it has to ascertain that there is no better z-optimal path $\mathbf{u}_z$. Besides this, the algorithm requires one call for every line segment in the cost curve. Hence, it needs $2K$-1 calls to the dynamic program in total.

Numerical accuracy is a problem for large alignments, particularly as M_1 grows larger. If a recombination does not reduce the mutation cost much, M_2 is still close to M_1 in

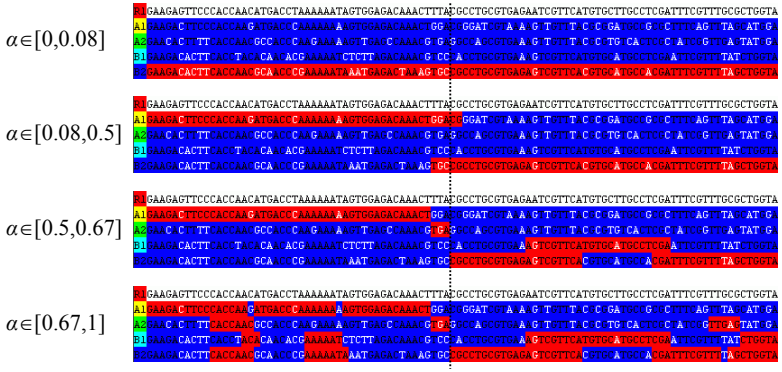


Figure 5.10: The parametric analysis allows to recover the optimal paths for all settings of α. Each set of α-optimal paths is valid for an interval of α, rounded here to two decimal places. As α gets larger, the α-optimal paths introduce more and more recombinations and the inferred recombination events become less significant. The α-optimal paths in the lowest row for example avoid mutations with respect to the query sequence by introducing recombinations and are usually not of interest.

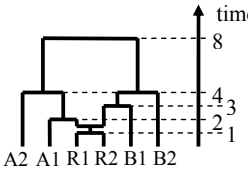
absolute terms. Therefore, the algorithm has to compute the intersection of two almost parallel lines corresponding to U_1 and U_2 and numerical instabilities can occur.

5.4.2.3 Parametric Analysis of the Example Alignment

The result of the parametric optimization for our example alignment is shown in Figure 5.9. Each sequence of the alignment was taken as the putative recombinant, in turn. The parametric cost curve for R1 shows that there are four sets of α-optimal paths that correspond to different intervals for α. Each linear piece and the corresponding set of α-optimal paths are characterized by a different amount of recombination and mutation cost. As expected, the curve corresponding to R1 shows a particularly high preference for recombination: $\alpha_1 \approx 0.077$ and $Savings_1 = 12$. Interestingly, the α-optimal paths with one recombination are also very robust and remain α-optimal up to $\alpha_2 = 0.5$. Figure 5.10 shows all α-optimal paths for the query sequence R1 and different settings of α.

5.5 The Connection Between the ARG and the α-Optimal Paths

The α-optimal paths are intuitive as they are the result of cost optimization. But it is unclear what we would expect as an α-optimal path if we use sequences as input simulated according to some ARG. The motivation for studying the expected α-optimal paths is manifold. First, we can draw conclusions from the α-optimal paths about the ARG underlying a sequence alignment. Second, we can check whether the α-optimal paths fit to our expectations if we simulate sequence alignments according to a given ARG. Finally, we



time	$E[Y_{100}]$	$E[X_{100}]$	$E[Y_{1000}]$	$E[X_{1000}]$
8	0.50	0.36	0.050	0.048
4	0.25	0.21	0.025	0.025
3	0.19	0.17	0.019	0.019
2	0.13	0.12	0.013	0.012
1	0.06	0.06	0.006	0.006

Figure 5.11: The example ARG from Figure 5.1 is shown at the left, where time is scaled such that there are 25 mutations per sequence from the root to a tip of the tree, on average. Only the analyses in section 5.5.4 retain sequence R2 in the alignment. A conversion table is shown at the right. The columns $E[Y_{100}]$ and $E[Y_{1000}]$ correspond to different sequence lengths N=100 and N=1000 and list the expected number of true mutations per site between two sequences that coalesce at a given time point. The columns $E[X_{100}]$ and $E[X_{1000}]$ list the expected number of observed mutations per site for the Jukes-Cantor mutation model. The expected number of true mutations per genome from the root to a tip of the tree is constant for all three scenarios. As N grows larger, $E[X_N]$ is a better estimate of $E[Y_N]$. Such a simple conversion table is only possible because all sequences are sampled at the same time point.

can deduce the expected $Savings_k$ values for simulated alignments and predict the strength of the recombination signal that Recco infers for a given ARG.

To keep the discussion limited, this section only conveys an understanding for the expected α-optimal paths and lacks some mathematical detail. For example, mutation models and the random process they describe are not introduced in detail. More information are provided in (Li 1997) and other books. Throughout the discussion it is also assumed that Recco uses Hamming distances for mutation and recombination cost. A more thorough analysis is left for future work.

5.5.1 Distance Correction and the Expected Number of Mutations

Given an ARG and a mutation model, we can simulate sequences of length N along the ARG. Branch lengths in the ARG are usually measured in terms of the expected number of mutations per site, and a larger N would result in more mutations per genome. To make comparisons possible, it often makes sense to keep the total number of mutations constant and scale the expected number of mutations per sites appropriately. Another restriction is that N must be large enough to represent the different segments in the ARG appropriately.

Starting with an arbitrary sequence at the root of the ARG, the mutation model generates random mutations and adds them to the sequences. If we use the infinite sites model, N=∞ and every mutation occurs at a different sequence position. The finite sites model allows recurrent mutations, such that several mutations can occur at the same position. If

we compare two sequences in this case, we may not always observe the true number of mutations that separate these sequences. In the following, distances in the ARG refer to the branch lengths in the ARG and therefore to the expected number of true mutations. Expected genetic distances, or expected distances in the sequences correspond to the expected number of observed mutations.

A particularly simple mutation model is the Jukes-Cantor model. The Jukes-Cantor correction (see Chapter 4 in (Li 1997)) predicts that

$$s(e) = -\frac{3}{4}\ln\left(1-\frac{4}{3}e\right), \text{ or equivalently } e(s) = \frac{3}{4}-\frac{3}{4}\exp\left(-\frac{4}{3}s\right) \tag{5.35}$$

where s is the true number of mutations per site and e is the observed number of mutations per site. There are distance corrections for other mutation models, too. Obviously, we cannot apply the distance correction to the observed mutations at a site, as e is either 0 or 1 in this case and e=1 leads to a negative logarithm. The Jukes-Cantor correction only makes sense in practice if the observed number of mutations per site is averaged over the whole sequence or at least over several sites.

Define $X_N(i,p)$ as the random variable that counts the number of *observed* mutations at position p between sequence i and the query sequence for simulated sequences of length N. Define $Y_N(i,p)$ as the random variable that counts the number of *true* mutations between the two sequences. $Y_N(i,p)$ is Poisson-distributed, as the waiting time for a mutation is exponentially distributed. The mean $E[Y_N(i,p)]$ is the distance between the two sequences in the ARG. The distribution of $X_N(i,p)$ is more complex. The mean of $X_N(i,p)$ depends on the distance between the two sequences in the ARG and on the mutation model. We can determine the mean of $X_N(i,p)$ for the Jukes-Cantor model as described above: $E[X_N(i,p)]=e(E[Y_N(i,p)])$ or equivalently $E[Y_N(i,p)]=s(E[X_N(i,p)])$. This estimates the distances in the ARG from the distances observed in the sequences. The distances in the ARG and the distances expected with the Jukes-Cantor model for the example ARG are shown in Figure 5.11.

5.5.2 Different Distance Estimates Lead to Different Analyses

If we assume Hamming distances for the mutation cost in Recco, $m(i,p)$ is distributed according to $X_N(i,p)$ and counts the number of observed mutations. The expected cost of an arbitrary path $\mathbf{u}$ is then

$$\begin{aligned} E\left[\varphi_\alpha(\mathbf{u},1,N)\right] &= E\left[(1-\alpha)R(\mathbf{u},1,N)+\alpha\sum_{p=1}^{N}X_N(u_p,p)\right] \\ &= (1-\alpha)R(\mathbf{u},1,N)+\alpha\sum_{p=1}^{N}E\left[X_N(u_p,p)\right] \end{aligned} \tag{5.36}$$

as the recombination cost is not a random variable, but a property of the path. Consequently, the dynamic program recovers the path $\mathbf{u}$ with the expected minimum cost if we set $m(i,p)=E[X_N(i,p)]$. The paths of expected minimum cost are also called α-optimal paths in the following, as the distinction is usually clear. It is important to realize that this

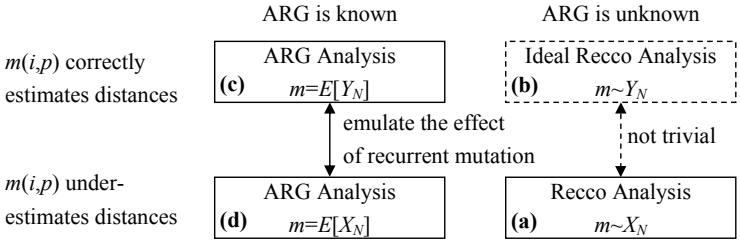


Figure 5.12: There are four different types of analyses. The right column refers to analyses that estimate branch lengths in the ARG from sequence data and are of practical use. We can transform the upper-left analysis into the lower-left analysis and vice versa as shown by the arrow. A similar transformation is not possible for the analyses on the right hand side. **(a)** Recco is based on cost optimization and does not correct for recurrent mutations. It systematically underestimates the distances in the ARG. **(b)** An ideal Recco analysis would correct for recurrent mutations and estimate the distances in the ARG accurately. **(c)** It is particularly intuitive to analyze the expected output of the ideal Recco analysis for a given ARG. **(d)** We can get an accurate estimate of the output expected from the Recco analysis if we emulate the missing distance correction for the branch lengths in the ARG.

approach does not necessarily recover the path **u** with the maximum likelihood to be α-optimal. But the differences are not very large in practice. For notational convenience, the indices i and p are dropped in the following, such that $m(i,p)=E[X_N(i,p)]$ is abbreviated as $m=E[X_N]$, for example. As before, the parametric analysis recovers all α-optimal paths ordered by increasing recombination cost.

An interesting alternative is to set $m=E[Y_N]$ and study the α-optimal paths if the mutation cost accurately estimates the distances in the ARG. In this case, it is actually not necessary to work with N columns in the dynamic program if Hamming distances are used for recombination cost. The values $E[Y_N(u_p,p)]$ only change at breakpoint locations in the ARG, and a large N introduces many identical columns in $m(i,p)$. However, it is suboptimal to introduce a recombination event between two identical columns, as it is either optimal to recombine before or after the two identical columns (see section 5.3.5 for a proof to a closely related problem). Identical columns can be consolidated into one column and the solution is identical for different N – at least if N is large enough to correctly represent the local trees in the ARG. This also allows recovering the α-optimal paths for an ARG manually without using the dynamic program. However, the analysis becomes quickly complicated if the ARG contains many recombination events.

There are four different types of analyses that are now possible, depending on the m used as input (see Figure 5.12). If the ARG is given, we can use $m=E[X_N]$ or $m=E[Y_N]$ in the dynamic program. This approach is called ARG analysis in the following. The first

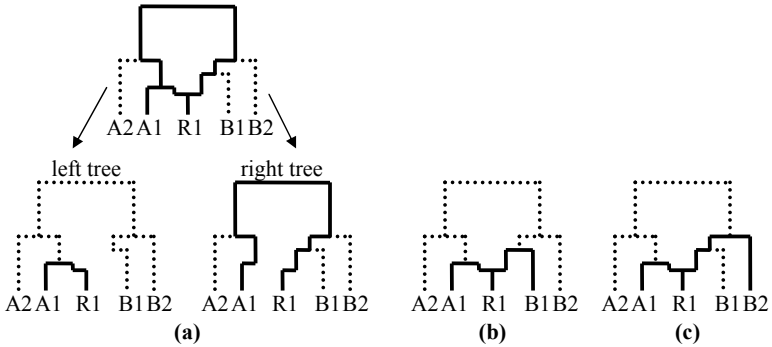


Figure 5.13: Different paths (or hypotheses) for R1 are visualized in the ARG. Dotted lines refer to parts of the ARG that do not contribute towards the cost of the path. **(a)** The α-optimal path for R1 that does not incur a recombination only covers A1. For the sake of clarity, the induced paths in the left and right tree of the ARG are shown here. The left part of the alignment does not contribute much mutation cost, but the right part does. **(b)** If one recombination is allowed, the α-optimal path for R1 covers A1 in the right and B1 in left part of the alignment. This is also the α-optimal path that we would ideally want to recover. The total mutation cost is lower than for (a), but the path involves a recombination. **(c)** The mutation cost increases only little if the path uses A1 and B2 to explain R1. It is difficult to decide between paths (b) and (c) if the mutation process introduces noise and it is not surprising that the α-optimal path for the example alignment uses B2 instead of B1.

choice $m=E[X_N]$ recovers the α-optimal paths expected for Recco and underestimates distances in the ARG. Alternatively, setting $m=E[Y_N]$ uses the correct distances in the ARG for the analysis. The effect of recurrent mutations vanishes as N grows larger, and both settings lead to the same result in the limit. In any other case, we have to estimate m from sequence data. As Recco does not perform any distance correction, m is distributed as X_N and consistently underestimates distances in the ARG. In an ideal scenario, Recco would use a distance correction on m, such that m is distributed as Y_N. Unfortunately, it is not trivial to implement a distance correction for Recco, as Recco requires distance estimates for every single column in the alignment.

5.5.3 ARG Analysis for the Example ARG

An ARG analysis based on $m=E[Y_N]$ is particularly intuitive, as m directly represents distances in the ARG. In some sense, an α-optimal path $\mathbf{u}$ predicts the sequence u_p closest to the query sequence for every position p. However, depending on α, the α-optimal path only allows for a limited number of recombinations and restricts the number of closest sequences to the query. The α-optimal path therefore corresponds to a hypothesis

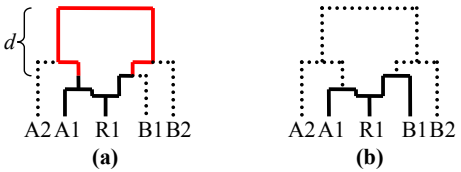


Figure 5.14: The two α-optimal paths (a) and (b) of Figure 5.13 are compared in more detail. Path (a) does not introduce a recombination, but forces the red detour in the ARG. The detour corresponds to the additional mutation cost M_2-M_1 of path (a) if $m=E[Y_{100}]$. The detour has a length of $2d=2\cdot0.25\cdot5/8\approx0.31$ expected true mutations per site. As the detour only covers the right half of the alignment, it increases the expected number of true mutations in path (a) by $d\approx50\cdot0.31\approx15.6$. Consequently, the detour corresponds to $Savings_1=15.6$ and $\alpha_1\approx1/(1+15.6)\approx0.06$, as we use Hamming distances for recombination and mutation. Hence, we expect to recover path (a) for $\alpha<0.06$ and path (b) for $\alpha>0.06$.

regarding the local structure of the ARG around the query sequence. The mutation cost of an α-optimal path can be visualized by following the branches in the ARG from the query sequence to the other sequences. Figure 5.13 shows the α-optimal paths for the example ARG that involve zero or one recombinations. Paths with more than one recombination do not lower the mutation cost and are not α-optimal for $\alpha<1$, as the minimum amount of mutation cost is bounded by the ARG. Paths in an ARG analysis are generally not α-optimal for $\alpha<1$ if they incur more recombination events than the number of recombination events reachable from the query sequence. A recombination event is reachable if there is some position p, such that the shortest path connecting the query and the closest sequence in the local tree for position p encounters the recombination event. Interestingly, there is a visual interpretation of α_1 and M_2-M_1 for the example ARG (see Figure 5.14).

The presented analysis shows the ideal scenario for Recco when distances are correctly estimated. It also identifies the different α-optimal paths that we hope to recover in reality. The values $\alpha_1\approx0.06$ and $Savings_1=15.6$ in the analysis are remarkably close to the values $\alpha\approx0.077$ and $Savings_1=12$ derived from the actual sequence alignment. The analysis also predicts that an α-optimal path for A1 should introduce a recombination for $\alpha>0.14$ (see Figure 5.15). Besides R1 and A1, B1 is the only other query sequence that should theoretically introduce a recombination in the α-optimal path because of the structure of the ARG. However, the expected value $Savings_1\approx3.13$ for B1 is so small that it does not matter in practice.

The result of the ARG analysis for $m=E[X_{100}]$ is very similar to the previous analysis. The distances that emulate the effect of recurrent mutations are given in Figure 5.12. The α-optimal paths are equal to the previous subsection for the example ARG, but their costs

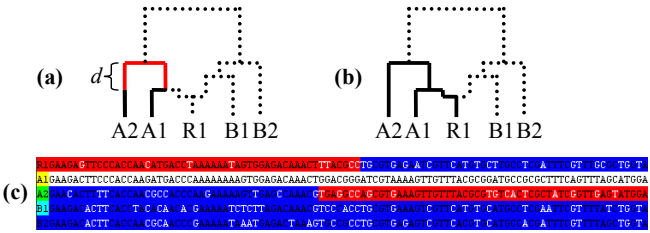


Figure 5.15: **(a),(b)** The same analysis as in Figure 5.14 is performed for the query sequence A1. The detour is much smaller and only increases the expected number of true mutations by d=50·2·0.25·2/8=6.25. Hence, path (a) is preferred over path (b) for α<0.14. Even though A1 is not a recombinant, we expect that an α-optimal path for A1 in a simulated alignment involves a recombination. **(c)** As expected, the α-optimal paths for the example alignment and α=0.2 correspond to path (b).

differ. However, for different ARGs, the α-optimal paths may differ between the two analyses. The expected number of mutations for path (a) and (b) in Figure 5.14 is M_1≈100·(0.5·0.12+0.5·0.36)=24 and M_2≈100·(0.5·0.12+0.5·0.17)=14, respectively. Hence, the expected values $Savings_1$≈24-14=10 and α_1≈0.09 are even closer to the values obtained from the example alignment.

5.5.4 Expected Savings for Different Scenarios

The $Savings_1$ value is central to the recombination analysis that Recco performs, as it compares the cost of α-optimal paths with no and at least one recombination event. This approach directly transfers to the analysis of ARGs as shown before. In the following, $Savings_1$ is compared for three different scenarios that are all based on the example ARG:

- **(S1)** the example ARG without sequence R2 and sequence length N=100.
- **(S2)** the example ARG without sequence R2 and sequences length N=1000. Time is scaled such that the average number of mutation events is the same as in (S1).
- **(S3)** the example ARG including sequence R2 and sequence length N=100

The theoretical analysis is augmented by simulating 1000 alignments for each scenario and computing $Savings_1$ for every alignment and query sequence. (S1) and (S2) differ only in the amount of recurrent mutations, but not in the expected number of true mutations. (S1) and (S3) show the effect on $Savings_1$ if a sequence R2 is closely related to the recombinant R1.

Table 5.1 shows the results of the theoretical and empirical analysis of $Savings_1$ for (S1) and (S2). The mean $Savings_1$ values measured in the simulated alignments are close to the expected $Savings_1$ values for R1, A1 and B1. The noise introduced by the random mutation process becomes particularly visible for query sequences A2 and B2. The ARG analysis ignores the noise and predicts $Savings_1$=0 for these query sequences, as it does not make sense to introduce any recombinations into the α-optimal paths. The empirical

Query sequence	ARG Analysis $m=E[Y_{100}]$	ARG Analysis $m=E[X_{1000}]$	Recco $m\sim X_{1000}$	ARG Analysis $m=E[X_{100}]$	Recco $m\sim X_{100}$
R1	15.6	14.9	14.0	10.0	9.46
A1	6.25	6.10	6.56	4.87	5.07
B1	3.13	3.04	4.44	2.34	3.64
A2	0	0	2.40	0	2.27
B2	0	0	2.65	0	2.49

Table 5.1: The expected $Savings_1$ value for different query sequences and analyses is shown for (S1) and (S2). The values for the Recco analyses are obtained from 1000 simulated alignments. The expected values are close to the empirical values and illustrate the accuracy of the ARG analysis.

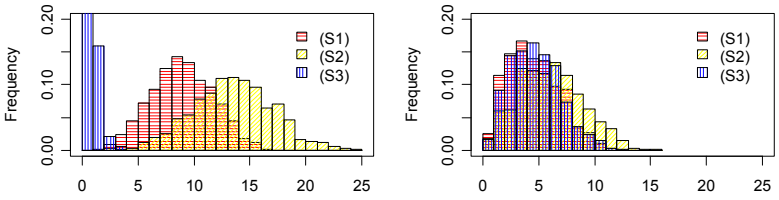


Figure 5.16: The histogram of $Savings_1$ differs for (S1), (S2) and (S3) and the different query sequences. **(a)** The effect of recurrent mutations becomes particularly visible for R1 when (S1) and (S2) are compared. $Savings_1$ is larger for N=1000 than for N=100 as there are less recurrent mutations. (S3) shows an important limitation of Recco: taking R1 as a query does not yield high $Savings_1$ values, as R2 is similar to R1. The blue histogram actually extends far beyond the display range and almost reaches a frequency of 0.8. **(b)** The histograms for A1 differ much less than for R1. The effect of recurrent mutation is less pronounced as A1 has a lower $Savings_1$ value. Comparing (S1) and (S3) shows that $Savings_1$ increases slightly if R2 is added to the alignment. The increase is expected, as the α-optimal path for A1 with one recombination can now choose between R1 and R2 and use the sequence that fits better. M_2 is therefore smaller and $Savings_1$ larger in this scenario.

$Savings_1$ value for A2 and B2 is larger than zero solely due to the noise introduced in the mutation process. Figure 5.16 compares the different distributions of $Savings_1$ for all scenarios and for the query sequences R1 and A1.

5.5.5 α as the Choice Between Resolution and Robustness

The ARG analysis enables predicting the output of Recco for alignments of sequences that evolved according to a given ARG and also clarifies the role of the parameter α for

Recco. Ideally, we would like to recover a path that predicts all recombination events that are reachable from the query sequence as such a path contains the most comprehensive information on the ancestry of the query sequence. These paths also do not predict any detour when mapped to the ARG. In other words, they simply choose the closest sequence to the query sequence for each position in the alignment. Given the ARG we can recover these paths by setting $m=E[Y_N]$ and solving the optimization problem for α close to one or alternatively by finding the closest sequence to the query sequence for every position in the alignment. But the branch lengths cannot be estimated reliably in practice and are subject to noise if we try to estimate branch lengths for small windows in the alignment that only cover a few nucleotides. The parameter α plays a role similar to the size of the sequence window. For example, one mutation is sufficient to cause a recombination in the α-optimal path if $\alpha>2/3$. Hence, setting α close to one picks up the noise from the mutation process and erroneously predicts recombination events.

In other words, the parameter α controls the noise that is acceptable for estimating the distances and determines the size of a detour required to justify a recombination. Setting α close to 0 smoothes the noise caused by the random mutation process, but also limits the resolution of the method to large detours. If we assume that $m=E[Y_N]$, the method cannot recover a recombination that corresponds to a detour which involves a mutation cost less than $(1-\alpha)/\alpha$. The age of the recombination also plays a role if $m=E[X_N]$ or if $m\sim X_N$. An old recombination does not cause the same $Savings_1$ value as a younger recombination with a detour of the same size. To summarize, α chooses between robustness regarding estimation of distance and resolution regarding the size of a detour.

It is interesting to compare the approach of Recco with sliding window-based methods as the size of the sliding window also selects the tradeoff between resolution and robustness. A large window size directly leads to more robust estimates of the branch lengths if there is no recombination inside the window. However, the resolution is limited by the size of the window, as recombinant regions that are smaller than the window size cannot be detected. In comparison, Recco does not limit the resolution by using a fixed window size, but by setting the amount of mutation that is required to justify a recombination (or detour) in the ARG.

5.6 Recombination Analysis Using the Parametric Cost Curve

The previous sections introduced the machinery for processing an alignment and recovering all α-optimal paths. Unfortunately, the amount of information is often bewildering and it is usually neither possible nor interesting to analyze the complete result manually. In the following, the analysis is extended quantitatively, such that the results can also be used in an automated procedure discerning recombinant from non-recombinant alignments. The final goal is to provide an approach for addressing the main questions in recombination analysis (see section 4.1). In particular, I propose an approach for detecting recombination in the alignment, for detecting the recombination breakpoints and the

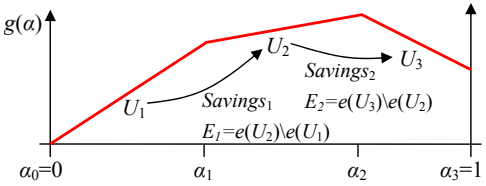


Figure 5.17: The parametric cost curve is annotated with several variables introduced in the text. The index *i* is dropped as only the cost curve for one sequence is shown. The arrows indicate transitions between different sets of α-optimal paths and are annotated with the recombination events that are introduced and with the corresponding Savings value.

recombinant sequences, and for finding the structure of a recombinant sequence. The approach also derives *p*-values for the output using the column permutation test (see section 4.2.4) and is the basis for the power study in the next chapter. To allow for a more concrete discussion, we define significant *p*-values as *p*-values below 5%.

The first step of the analysis is to collect the full information of the parametric analysis into an ordered list of recombination events. The list is already helpful by itself and allows for a much easier manual analysis of the recombination signal in the alignment. The analysis proceeds by computing the parametric cost curve for column permutations of the alignment, such that several *p*-values for detecting recombination can be derived. Besides the Savings value, there are several other features for detecting recombination that were described in the publication of Recco (Maydt and Lengauer 2006), but performed not as well for recombination detection.

5.6.1 The List of Recombination Events

The parametric cost curve is particularly attractive as a starting point for recombination analysis, as it summarizes all different recombination hypotheses and provides a lot of information on the ancestry of the query sequence. An automated analysis of an alignment usually requires taking each sequence in the alignment as a query, in turn. Hence, we introduce an additional index *i* for the parametric cost curve to identify the query sequence that produced the data. To be more specific, define U_{ki} as the set of α-optimal paths corresponding to the *k*-th segment of the parametric cost curve taking sequence *i* as a query. Define $Savings_{ki}$ as the Savings value corresponding to the transition from U_{ki} to $U_{k+1,i}$ (see Figure 5.17) and σ_i as the upper bound for $Savings_{1i}$. A recombination event (i,j,j',p) is defined by the query sequence *i*, the location of the recombination breakpoint *p* and the sequences *j* and *j'* left and right of the breakpoint. A path $\mathbf{u}$ then defines a set of recombination events $e(\mathbf{u},i)=\{(i,u_p,u_{p+1},p)|u_p\neq u_{p+1}\}$. Similarly, we can define the set of recombination events for a set of paths *U* as

$$e(U,i)=\bigcup_{\mathbf{u}\in U} e(\mathbf{u},i)\,. \tag{5.37}$$

It is important to understand that $e(\mathbf{u},i) \cap e(\mathbf{v},i)$ can be empty for some $\mathbf{u},\mathbf{v} \in U_{ki}$. In other words, two paths $\mathbf{u},\mathbf{v} \in U_{ki}$ do not necessarily have recombination events in common.

If the parametric cost curve is processed from left to right, it orders the U_{ki} and the corresponding recombination events $e(U_{ki},i)$ by decreasing Savings value. The paths in $U_{k+1,i}$ introduce the recombination events $E_{ki}=e(U_{k+1,i}) \setminus e(U_{ki})$. Hence, we can refer to $Savings_{ki}$ as the Savings value corresponding to E_{ki}, the set $U_{k+1,i}$ or to the recombination events introduced therein. To be more specific, we can define a function

$$Savings(x) = \max\left(\{Savings_{ki} \mid x \in e(U_{ki},i)\} \cup \{0\}\right) \tag{5.38}$$

for all recombination events x and refer to $Savings(x)$ as the Savings value of the recombination event x. We can then construct a list L containing all recombination events $\{e(U_{ki},i) | i,k\}$ ordered by decreasing Savings value. Alternatively, it is also possible to construct an ordered list L_i of recombination events for every query sequence i.

5.6.2 Interpreting the List of Recombination Events

It is usually convenient to compile the list of recombination events for the alignment or for a sequence and then examine each recombination event manually in the order of decreasing Savings value (see Figure 5.18). Recco does not list the source or destination sequence in the list of recombination events. However, selecting a recombination event in Recco quickly visualizes the associated α-optimal paths U_{ki} and zooms to the region surrounding the recombination breakpoint. The source and destination sequences are then easily recovered. This approach is particularly attractive for larger alignments, as it is usually not possible to visualize the full alignment in this case. As an alternative, the user can inspect the α-optimal paths U_{ki} ordered by increasing k manually, but then has to scroll to obtain the complete picture.

Recombination events with a small Savings value are usually not of interest, as the α-optimal paths corresponding to small Savings values are susceptible to noise by random mutations. There are two approaches implemented in Recco. The user can specify a cutoff ad-hoc and only keep the recombination events x that satisfy, for example, $Savings(x)>5$. Alternatively, Recco can also determine a cutoff based on column permutations and only present recombination events that are significant (see section 5.6.4). It is often necessary to combine both approaches in practice, as the p-values can be significant even for small Savings values.

We have to be careful about interpreting the recombination events in the sets U_{ki} and in the lists L and L_i. The recombination hypotheses U_{ki} for increasing k are ordered by an increasing amount of recombination cost. But there is usually no α-optimal path that contains several recombination events of the list. A simple observation is that different α-optimal paths for the same α can be completely unrelated. For example, the recombination events (a) in Figure 5.18 are all mutually exclusive and refer to different α-optimal paths in U_{2i}. There is no α-optimal path that contains two recombination events of (a). However, there is another effect that is counterintuitive and much harder to interpret. As

	i (query)	*j* (source)	*j'* (dest)	*p* (breakpoint)	Savings	Alignment *p*-value*	Sequence *p*-value*
(a)	R1	A1	B2	45, 48-50	12	0.0001	0.0001
(b)	A1	R1	A2	45-46, 49-52	7	0.0611	0.0358
(c)	A2	B2	A1	19-24	3	0.9549	0.4677
	A1	A2	B2	72-74	2	1.0000	0.9872
	A1	B2	A2	95	2	1.0000	0.9872
	…	…	…	…	…	…	…

* *p*-values are based on 10.000 permutations

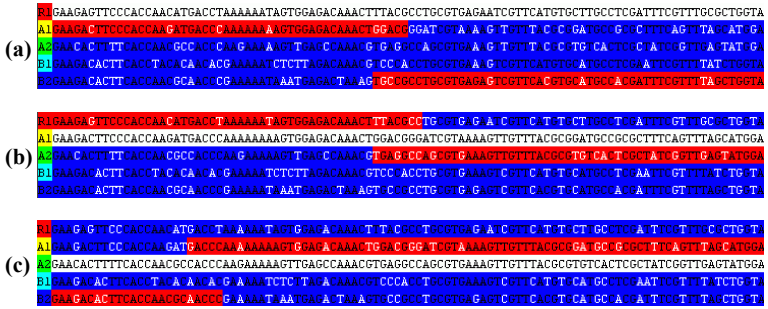
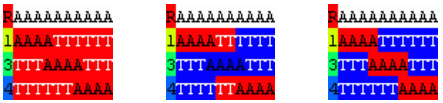


Figure 5.18: Shown are the first five elements of the recombination event list L for the example alignment, along with the corresponding α-optimal paths. The list shows recombination events in the same row if they only differ by the position of the breakpoint. From the ARG, we only expect to recover the first two recombination events in the list. The third recombination in the list is due to noise. The p-values are in concordance with our expectations: **(a)** is a highly significant recombination event, **(b)** is a borderline case and **(c)** is not significant anymore. The computation of the p-values and their interpretation is explained later in section 5.6.3.



R_1=0 M_1=6 **(a)** R_2=1 M_2=2 **(b)** R_3=2 M_3=0 **(c)**

Figure 5.19: The α-optimal paths are shown by an increasing amount of recombination. As the number of recombinations increase, recombination events introduced earlier may become obsolete and are not part of an α-optimal path anymore. The recombination events introduced in (b) are not encoded in (c) anymore.

we recover recombination hypotheses with an increasing amount of recombination, we might expect that the first recombination event is also part of a hypothesis with more recombination events. For example, the list in Figure 5.18 correctly suggests that some α-optimal path for query A1 is composed of the second, the fourth and the fifth recombination event and the predicted path is R1, A2, B2, A2, with breakpoints occurring at the positions given in the list. But there are cases where this heuristic fails and a recombination event at the top of the list is replaced by other recombination events later in the list. To be more specific, it does not hold in general that $e(U_{1i}) \subseteq e(U_{2i}) \subseteq \ldots \subseteq e(U_{Ki})$ (see Figure 5.19). This effect usually occurs if the parental sequences differ between the α-optimal paths. For example, the paths in U_{2i} may predict a recombination event x from sequence i to j at some position p. If x is not part of a path in U_{3i}, all paths in U_{3i} usually use parental sequences other than i and j at position p. Hence, an unstable recombination event also points to a problem of estimating the parental sequences correctly. Although these cases are not too frequent in practice, they present a challenge regarding the analysis of the recombinant structure with Recco. As explained before, we ideally want to recover a robust α-optimal path corresponding to a Savings value as low as possible. But if there is a recombination event with a large Savings value that becomes obsolete for lower Savings values, it is unclear whether the recombination event should be part of the desired α-optimal path. Even worse, Recco computes a small p-value for the recombination events at the top of the list, even though these events may become obsolete as more and more recombination events enter the α-optimal path. If there is no such recombination event, we can trust the ordering of events and know that recombination events corresponding to higher Savings values are more likely to represent true recombination events.

5.6.3 *P*-values for Recombination

$Savings_{1i}$ is a good indicator for detecting recombination in a query sequence i, as it measures the benefit of a recombination in the query. But we cannot rely on its absolute value for identifying recombination in general. The value of $Savings_{1i}$ depends on parameters of the alignment, such as the genealogy, sequence diversity or the sequence length (see Figure 5.16). Fortunately, the column permutation test provides a simple approach to derive p-values for the absolute $Savings_{1i}$ values. Computing $Savings_{1i}$ for many column permutations of the original alignment provides a good estimate of the distribution of $Savings_{1i}$ expected under the null hypothesis of no recombination. The p-value then estimates the probability of observing a value as large as $Savings_{1i}$ by chance in the original alignment and quantifies whether a sequence benefits significantly from introducing recombination into the α-optimal path. The p-value for $Savings_{1i}$ therefore can be used to detect whether sequence i is a recombinant.

The question about recombination in the alignment can be approached in a similar fashion. First, take each sequence of the alignment as a query, in turn, and compute the $Savings_{1i}$ values for each query. Then, the maximum of $Savings_{1i}$ over all query

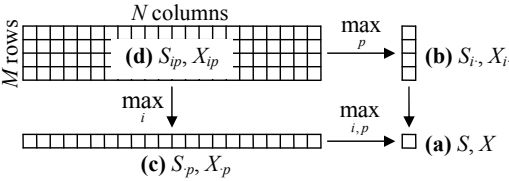


Figure 5.20: The values S_{ip} correspond to an array of the same dimension as the alignment. The maximum of S_{ip} over different dimensions can be used to obtain p-values for different purposes. The numbering (a) to (d) is the same as in the text describing the p-values below.

sequences i is a good indicator for recombination in the alignment, and the column permutation test applies analogously for computing p-values for recombination in the alignment. Methods for computing p-values at a certain position p are discussed in the following.

5.6.3.1 The Recombination Profile

The recombination event list summarizes a lot of information on the parametric cost curve. A different representation is sometimes more suitable for computing p-values, particularly for detecting recombination breakpoints. The basic idea is to compute a recombination profile S_{ip} for every query sequence i that stores the maximum Savings value for a recombination event at position p. To be more specific, define the recombination profile as

$$S_{ip} = \max\left\{Savings(x) \mid x = (i, j, j', p) \in L_i\right\}. \tag{5.39}$$

Now, let X_{ip} be the random variable describing the S_{ip}'s for column permutations of the original alignment. It is particularly intuitive to think of S_{ip} as an array of the same dimension as the alignment (see Figure 5.20). Processing a column permutation of the original alignment results in an array of the same dimension, and the distribution of X_{ip} can be estimated easily. For notational convenience, we abbreviate the maximum of X_{ip} and S_{ip} regarding i or p in the following:

$$X_{\cdot p} = \max_i\left\{X_{ip}\right\}, \quad X_{i\cdot} = \max_p\left\{X_{ip}\right\}, \quad X = \max_{ip}\left\{X_{ip}\right\}. \tag{5.40}$$

S_{ip} is handled accordingly and satisfies additionally

$$S_{i\cdot} = Savings_{1i} \text{ and } S = \max_i\left\{Savings_{1i}\right\}. \tag{5.41}$$

5.6.3.2 *P*-values for Recombination in the Alignment, the Sequence and at a Breakpoint

As depicted in Figure 5.20 there are four different approaches to maximizing S_{ip} over different dimensions. Each approach results in different p-values with different interpretations:

(a) *P*-value for Recombination in the Alignment. As described before, the *p*-value for recombination in the alignment tests whether the maximum Savings value in the alignment is significantly large:

$$\Pr(X \geq S) = \Pr\left(\max_{i,p}\{X_{ip}\} \geq \max_{i,p}\{S_{ip}\}\right). \tag{5.42}$$

(b) *P*-value for Recombination in Sequence *i*. We can also derive a *p*-value that measures whether the query sequence *i* significantly benefits from introducing a recombination into the α-optimal path:

$$\Pr(X_{i\cdot} \geq S_{i\cdot}) = \Pr\left(\max_{p}\{X_{ip}\} \geq \max_{p}\{S_{ip}\}\right). \tag{5.43}$$

A significant *p*-value does not necessarily imply that sequence *i* is a recombinant, but may only indicate that a closely related sequence is a recombinant. For example, we expect to detect sequence A1 in the example alignment as a recombinant (see section 5.5.3). However, it is in general difficult to separate recombinant sequences from closely related non-recombinant sequences (see section 4.1).

(c) *P*-value for a Recombination Breakpoint at Position *p*. Computing the maximum Savings value for each position *p* in the alignment allows for a simple approach to obtain the *p*-value for a recombination breakpoint at position *p*:

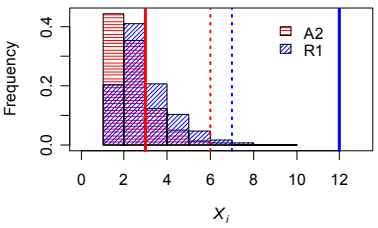
$$\Pr(X_{\cdot p} \geq S_{\cdot p}) = \Pr\left(\max_{i}\{X_{ip}\} \geq \max_{i}\{S_{ip}\}\right). \tag{5.44}$$

(d) *P*-value for a Recombination Breakpoint in Sequence *i* at Position p. Finally, we can also directly compare the S_{ip} values with the distribution expected under the null hypothesis and get the *p*-value for a recombination breakpoint in sequence *i* at position *p*:

$$\Pr(X_{ip} \geq S_{ip}). \tag{5.45}$$

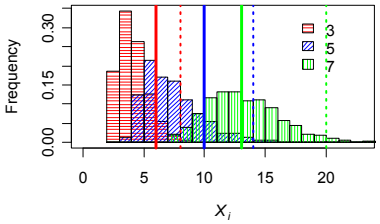
The different *p*-values presented above are related to each other, but may still result in very different predictions regarding the recombination signal in the alignment. For example, it is possible that the *p*-value for recombination in the alignment does not detect any significant recombination, but that a *p*-value for recombination in a query sequence is significant. There are only two factors that determine the *p*-values and may cause these differences: the observed values S and $S_{i\cdot}$ in the right side of $\Pr(\cdot)$ and the random variables X and $X_{i\cdot}$ in the left side of $\Pr(\cdot)$

Now, let us focus on the random variables first and study how the distribution of X_{ip} varies for different *i* and *p*. X_{ip} only depends on the distribution of columns in the alignment,



	$E[X_{i\cdot}]$	95%	σ_i	M_{1i}
R1	3.40	7	18	24
A2	2.75	6	15	26

(a)



	$E[X_{i\cdot}]$	95%	σ_i	M_{1i}
3	4.53	8	50	79
5	7.76	14	98	150
7	13.73	20	128	217

(b)

Figure 5.21: The plots visualize the distribution of $X_{i\cdot}$ for different query sequences i. The vertical lines visualize the Savings value for the query sequence in the original alignment, and the dotted vertical lines show the 95% quantile of the corresponding distribution. The table below the plot lists some values in detail, such as the mean and the 95% quantile of $X_{i\cdot}$ and the bound σ_i. **(a)** The example alignment without sequence R2 was used as an input. The two queries show slightly different Savings histograms for the column permutations. $Savings_1$ of R1 is clearly significant, while $Savings_1$ of A2 is not. **(b)** The effect that the distribution of $X_{i\cdot}$ depends on the query sequence i is more pronounced for alignments with a higher diversity. The histogram displays the result of 1000 column permutations of scenario (S3) in section 7.1.2. For the matter of this plot, it is only relevant that all three query sequences are not involved in a recombination, but differ in the bound σ_i and M_{1i}. Both values also are highly correlated with $E[X_{i\cdot}]$ and the 95% quantile. All p-values for recombination in a sequence are not significant here, but it is clear that a significant p-value for sequence 3 does not require a Savings value as high as for sequence 7.

as X_{ip} is obtained from column permutations of the original alignment. In other words, the ordering of columns in the alignment is immaterial and only the frequency for each column state is relevant. Figure 5.21 compares the distribution of $X_{i\cdot}$ for different query sequences and alignments and shows that $X_{i\cdot}$ can vary greatly for different i. It is important to realize that Figure 5.21 is very different from Figure 5.16: the first visualizes the distribution of Savings for 1000 permutations of one alignment, while the latter visualizes the distribution of Savings for 1000 alignments simulated according to the same ARG. Interestingly, the histograms in Figure 5.21 suggest that $X_{i\cdot}$ is approximately Poisson-distributed. The parameter of the distribution apparently depends on the bound σ_i for the Savings value (see section 5.4.2.1) and on M_{1i}, but is also influenced by other factors.

Figure 5.22 shows that $X_{\cdot p}$ strongly depends on p. High Savings values are much more likely at the center of the alignment than at the boundaries, because a recombination at

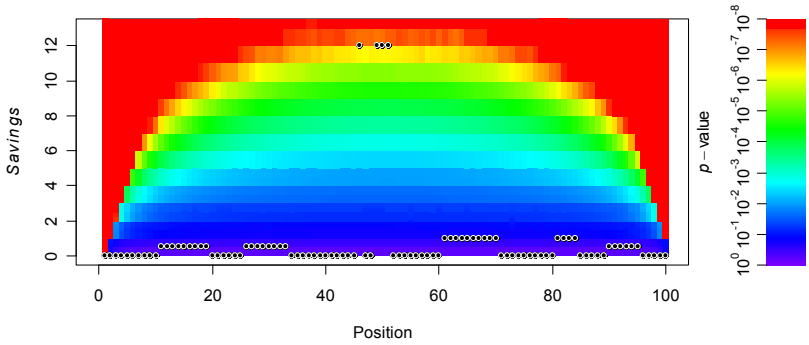


Figure 5.22: 10^8 column permutations of the example alignment were processed to approximate the distribution of $X_{\cdot p}$ for all p. The background color encodes the probability of observing a certain Savings value by chance. The probability of observing a large Savings value is much higher at the center than at the boundaries of the alignment. The black dots visualize the recombination profile $S_{\cdot p}$ of the original alignment. Each black dot corresponds to the p-value color directly below the dot. The recombination breakpoint in the middle of the alignment is highly significant and has a p-value of about 10^{-6}. All other breakpoints have p-values greater than 0.29 and are not significant.

the boundary exchanges fewer nucleotides and cannot achieve high Savings values easily. The same effect also occurs for X_{ip} if we compare different positions p (not shown). The distribution of X_{ip} therefore depends on the query sequence i and on the position p. The mean of X_{ip} is small if the position p is close to the boundary or if σ_i is small, for example because the alignment contains a sequence similar to the query sequence i.

We can now explain in more detail how the p-values (a)–(d) differ. Define j as the sequence and q as the position where S_{ip} achieves the maximum. Then it follows that $S_{jp}=S_{\cdot p}=S_{j\cdot}=S$ and we can compare the four p-values for sequence j and position q. For every realization x_{ip} of X_{ip} it holds that

$$x_{ip} \le x_{i\cdot} \le x \quad \text{and} \quad x_{ip} \le x_{\cdot p} \le x \tag{5.46}$$

and therefore we know that

$$\begin{aligned} \Pr(X_{ip} \ge S) &\le \Pr(X_{i\cdot} \ge S) \le \Pr(X \ge S) \\ \Pr(X_{ip} \ge S) &\le \Pr(X_{\cdot p} \ge S) \le \Pr(X \ge S) \end{aligned}. \tag{5.47}$$

In other words, the p-value (d) for a recombination breakpoint in sequence j at position q is at most as large as the p-value (c) for a recombination at position q or the p-value (b) for a recombination in sequence j. The p-value (a) for recombination in the alignment is at least as large as any of the other p-values. The p-value for recombination in a sequence can therefore be significant even though the p-value for recombination in the alignment is not, simply because the random variables X and $X_{i\cdot}$ differ.

Even though the *p*-value for recombination in the alignment is the most conservative, all four *p*-values control the false detection rate correctly. If we consider *p*-values below 0.05 as significant, we expect that less than 5% of the significant *p*-values detect a false recombination signal. The empirical probability is usually well below 5%, as the random variables X_{ip}, $X_{\cdot p}$, $X_{i\cdot}$, X all have a discrete domain (not shown here). However, there is more than one *p*-value for (b)–(d) and testing all of these *p*-values for significance leads to a multiple testing problem. The multiple testing problem is usually negligible for *p*-value (b), as there are often less than 20 sequences in an alignment and the detected events can still be validated manually. But the *p*-values (c) and (d) refer to positions in the alignment. We already expect five spurious recombination breakpoints for the example alignment with N=100. Therefore, we would have to adjust for multiple testing, e.g. using the Bonferroni correction or false discovery rate control (Benjamini and Hochberg 1995). The Bonferroni correction multiplies *p*-values by the number of conducted tests before they are compared with the significance level, or alternatively divides the significance level by the number of tests. A significance level of 0.05 therefore corresponds to a significance level 0.0005 after correcting for 100 tests. Correcting for 100 tests therefore requires computing more accurate *p*-values and increases the number of permutations and the computation cost by 100 times. But adjusting for multiple testing also renders many *p*-values insignificant and decreases the power of detecting breakpoints. In summary, *p*-values for recombination in the sequence and/or at a position are not only less conservative than the *p*-value for recombination in the alignment, but also entail a multiple testing problem. The *p*-values for detecting the breakpoints are rarely useful in practice, and a better approach is presented in the next subsection.

5.6.4 Estimating the Recombinant Structure of the Query Sequence

Given an alignment, we can determine whether there is recombination in the alignment and which sequences are most likely recombinants. However, the *p*-values for detecting the recombination breakpoints are problematic, as they entail a multiple testing problem. An alternative is presented in the following: instead of testing every position for a significant breakpoint, we can determine the largest α that is still robust regarding the noise introduced by the mutation process. The α-optimal path is then a good estimate of the recombinant structure of the query sequence and implicitly predicts recombination breakpoints.

5.6.4.1 A Robust Savings Threshold

How can we estimate the recombinant structure of a query sequence i? As described in section 5.5.5, the parameter α and the corresponding Savings value act as a smoothing parameter. If the smoothing is too strong, Recco does not correctly estimate the recombinant structure. But if the smoothing is too weak, Recco may erroneously introduce recombination events into the α-optimal paths. It is often better to be conservative and miss

a true recombination event instead of detecting a spurious recombination event. Thus, the smoothing should rather be too strong than too weak.

Given the random variable $X_{i\cdot}$ it is easy to find a threshold T_i such that $\Pr(X_{i\cdot} \geq T_i) \leq 0.05$ for each query sequence i. In other words, T_i is the 95% quantile of $X_{i\cdot}$ and we expect to observe a Savings value as extreme as T_i only in 5% of the permuted alignments. Define $\tau_i = 1/(1+T_i)$ as the α value corresponding to T_i. Then, the τ_i-optimal paths erroneously contain one or more recombination events only in 5% of the cases. However, a τ_i-optimal path may recover more than one falsely detected recombination event in this case. As an alternative to T_i, we can also find a threshold T for the whole alignment. T then satisfies $\Pr(X \geq T) \leq 0.05$ and we can define $\tau = 1/(1+T)$. The probability that a τ-optimal path infers at least one recombination event erroneously for any of the query sequences is only 5%, but it may infer more than one spurious recombination event just like the τ_i-optimal paths before. Hence, it is a concern that more than 5% of the detected recombination events are spurious even if we set the threshold or significance level to 5%.

T_i and T work best for different application scenarios, similar to the p-values for recombination in a sequence and in the alignment. T_i estimates the noise of the mutation process for each query sequence separately. As the mutation process introduces more noise if the query is distantly related to all other sequences in the alignment, T_i is larger in such a case. This reflects the fact that it is harder to estimate the ancestral relationship correctly. The smoothing parameter T_i therefore adapts the sensitivity of Recco regarding recombination detours to the query sequence. However, T_i is problematic if it is very small, as the smoothing may detect recombination events based only on a few mutations. As these recombination events are not very informative, it is usually better to enforce a lower bound on T_i, such as $T_i \geq 5$. The examples in section 5.6.4.3.4 also confirm that the smoothing may infer too many recombination events if T_i is very small. In contrast to T_i, T estimates the maximal smoothing necessary for any sequence in the alignment. In analogy to the p-value for recombination in the alignment, T controls the false detection rate better than T_i and avoids the problem associated with testing several sequences for recombination. T is also conservative in practice and does not infer many spurious recombination events. Nevertheless, T_i is more appropriate for analyzing recombination in a single query sequence.

5.6.4.2 *P*-values for Recombination Events

The random variables $X_{i\cdot}$ and X can also be used to assign a p-value to each Savings value and each recombination event. Recco refers to $\Pr(X_{i\cdot} \geq Savings(e))$ as the sequence p-value of a recombination event e and to $\Pr(X \geq Savings(e))$ as the alignment p-value of e (see Figure 5.18 for the recombination event list including these p-values). For every recombination event $e=(i,j,j',p)$, we know that the alignment p-value is larger than the sequence p-value, and that the sequence p-value is larger than the p-value for a recombination breakpoint in sequence i at position p. Furthermore, large Savings values correspond to

small p-values, and the sequence (or alignment) p-values impose the same ordering as the Savings values for the recombination events in the list L_i (or L). We also know that all recombination events e with $Savings(e) \geq T_i$ (or $Savings(e) \geq T$) have a significant sequence (or alignment) p-value. The thresholds T_i and T therefore define the cutoff in the recombination event list that separates significant from insignificant recombination events. As described before, a problem occurs for the recombination event list if some recombination events are unstable and become obsolete for low Savings values. In this case, it is usually better to rely on T_i or T as a smoothing parameter, recover the corresponding τ_i or τ-optimal path and regard the recombination events therein as the correct estimate.
The p-values for the recombination events also allow for computing a cutoff in the lists L_i and L. Suppose that L_i contains the recombination events e_1, e_2, … . Hence, we can test $\Pr(X_i \geq Savings(e_1)) \leq 0.05$, $\Pr(X_i \geq Savings(e_2)) \leq 0.05$, … until a test fails and then return all recombination events that were accepted. The result is identical to returning all recombination events e with $Savings(e) \geq T_i$. The analogous approach for the list L recovers all recombination events e with $Savings(e) \geq T$.

5.6.4.3 Is the Savings Threshold Conservative?

Suppose that the Savings threshold T_i is computed for alignments without recombination. We know that the τ_i-optimal path inserts one or more recombination events with a probability of only 5%. But it is unclear whether a τ_i-optimal path would introduce more than one recombination events in such a case. In other words, it may happen that one falsely detected recombination event causes additional false detections, such that the estimate is very wrong. Hence, the question for non-recombinant alignments is: what is the probability of inferring a second recombination event if the first recombination event is significant? A similar question also occurs for alignments containing recombination: do we infer spurious recombination events besides the true recombination events and if we do, how many?

5.6.4.3.1 The Parametric Cost Curve for Column Permutations

To keep the discussion limited, we assume that Hamming distances are used as recombination cost. Furthermore, we only study one sequence at a time such that we do not need to introduce a sequence index for the elements of the parametric cost curve. The parametric cost curve corresponding to sequence i and to T_i then defines the set of paths U_k for $k=1,\ldots,K$, the Savings values $Savings_1 > \ldots > Savings_{K-1}$, and the set of recombination events $E_k = e(U_{k+1}) \backslash e(U_k)$. We can also compute these values for a column permutation of the alignment. Define $y_1 > \ldots > y_{K'-1}$ as the Savings values for a column permutation, where K' is not necessarily equal to K and refers to the number of segments in the parametric cost curve. It is simple to construct a list of the same length from the y_k's, though, by either discarding superfluous values y_k or by adding artificial values $y_k=0$. Hence, we can define random variables $Y_1,\ldots,Y_K$ that describe the distribution of the y_k's, where Y_1 is

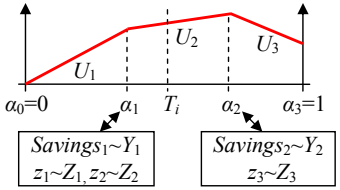


Figure 5.23: A parametric cost curve is shown for sequence i and annotated with the variables introduced in the text. $Savings_1$, z_1 and z_2 are all equal and correspond to α_1. The first transition from U_1 to U_2 introduces two recombination events as z_1=z_2. Consequently, the threshold T_i accepts both, z_1 and z_2. This case illustrates that more than one (potentially false) recombination event can be introduced simultaneously. The same effect can occur for $Savings_k$ as several Savings values can be larger than T_i and more than one Savings value is (potentially falsely) accepted simultaneously.

identical to the random variable $X_{i\cdot}$ introduced before (see Figure 5.23). The threshold T_i then determines the maximal c such $Savings_c \geq T_i$ and classifies the recombination events in $E_1 \cup \ldots \cup E_c$ as significant. We can also define $Savings_1$, $Savings_2$, … and Y_1, Y_2, … for the alignment by maximizing the corresponding $Savings_k$ and Y_k over all query sequences in the alignment. These values are referred to as $Savings_k$ and Y_k of the alignment in the following.

The previous analysis does not take into account that each Savings value may correspond to two or more recombination events, though. Remember that $Savings_k=(M_k-M_{k+1})/(R_{k+1}-R_k)$ and therefore that each of the $R_{k+1}-R_k$ recombination events in an α-optimal path eliminates $Savings_k$ in mutation cost. Hence, we can alternatively define $z_1,\ldots,z_J$ instead of $Savings_1,\ldots,Savings_{K-1}$, such that each z_j corresponds to a single recombination event in an α-optimal path. For example, if $Savings_1$=5 and R_2-R_1=2 then z_1=5 and z_2=5. It follows that

$$\sum_{k=1}^{K-1} Savings_k \left(R_{k+1} - R_k\right) = \sum_{j=1}^{J} z_j \,. \tag{5.48}$$

Define the random variables $Z_1,\ldots,Z_J$ such that they represent the distribution of $z_1,\ldots,z_J$ for column permutations of the alignment. As above, we can also define Z_j and z_j for the alignment by maximizing over the corresponding values for all sequences in the alignment. As z_j (or $Savings_k$) is a realization of Z_j (or Y_k) we do not distinguish between z_j (or $Savings_k$) and any realization of Z_j (or Y_k) in the following.

The following inequalities result from the definition of z_j and Z_j and the same inequalities are also true if we substitute z_j by $Savings_k$ and Z_j by Y_k:

1. $z_{j-1} \geq z_j$ for all j and all realizations z_j of Z_j.
2. it follows directly that $\Pr(Z_{j-1} \geq T_i \cap Z_j \geq T_i) = \Pr(Z_j \geq T_i)$
3. and hence $\Pr(Z_{j-1} \geq T_i) \geq \Pr(Z_{j-1} \geq T_i \cap Z_j \geq T_i) = \Pr(Z_j \geq T_i)$ for all j.

We can also derive inequalities relating z_k and $Savings_k$ for all $k \leq K$:

4. By definition we know that $z_k \geq Savings_k$ for all k.
5. It directly follows that $\Pr(Z_k \geq T_i) \geq \Pr(Y_k \geq T_i)$ for all k.

5.6.4.3.2 The Expected Number of Falsely Detected Recombination Events

We can now come back to the initial questions and describe them in more detail. Given the notation above, we would like to know the expected number of $Savings_k$'s or z_j's that are erroneously significant. We refer to the first as Err_{Sav} and to the latter as Err_z. Err_{Sav} or Err_z are lower for alignments with recombination than for alignments without recombination, as the large Savings values of the true recombination events lowers the Savings value of falsely detected recombination events (see next section). Consequently, we only study alignments without recombination in the following, such that Err_{Sav} and Err_z are the expected number of $Savings_k$'s and z_j's that are significant, respectively. We define Err_{Sav} and Err_z as

$$Err_{Sav} = \sum_{k=1}^{K} \Pr(Y_k \geq T_i) \tag{5.49}$$

$$Err_z = \sum_{j=1}^{J} \Pr(Z_j \geq T_i) \tag{5.50}$$

where $\Pr(Z_j \geq T_i)$ is the probability of falsely detecting j or more recombination events. It then follows that $Err_{Sav} \leq Err_z$ because $K \leq J$ and $\Pr(Y_k \geq T_i) \leq \Pr(Z_k \geq T_i)$ for all $k \leq K$. If Err_{Sav} or Err_z are much larger than our chosen significance level of 5%, we know that the threshold T_i often leads to several falsely detected recombination events in the τ_i-optimal path. Studying Err_z is intuitive from a user's perspective, as it determines the expected number of falsely detected recombination events. However, studying Err_{Sav} also makes sense, as the Savings value inherently controls the number of recombination events in the α-optimal paths. For example, if we erroneously accept $Savings_1$, we cannot avoid that the corresponding α-optimal paths may introduce more than one recombination event simultaneously. It is already very reassuring if Err_{Sav} is not much larger than 5%, as we then know that it is infrequent that more than one Savings value is falsely accepted in an alignment. In other words, even if the τ_i-optimal path infers too many recombination events, the α-optimal paths corresponding to the next larger Savings value usually do not. Hence, we would like to show that it is very rare that two or more $Savings_k$ are larger than the threshold T_i, as we then know that T_i consistently picks conservative τ_i-optimal paths.
But when is Err_{Sav} or Err_z much larger than 5%? We focus on Err_z in the following as it is more intuitive in practice, but the same results transfer to Err_{Sav} analogously. Recco falsely detects more than one recombination event if there are several z_j's that are larger or equal to the threshold T_i and Err_z is large if $\Pr(Z_j \geq T_i)$ does not decrease rapidly with increasing j. For example, assume that Z_1 and Z_2 are positively correlated and that any realization z_1 and z_2 satisfies $z_2 = z_1$. In this case, z_2 is significant if z_1 is significant and we detect either zero or at least two false recombination events in the alignment.

We can quantify this effect in more detail by studying $\Pr(Z_j \geq T_i | Z_{j-1} \geq T_i)$, the probability of detecting a j-th recombination event if a $(j-1)$-th recombination event has been detected. $\Pr(Z_j \geq T_i | Z_{j-1} \geq T_i)$ is particularly intuitive as it captures the dependency between Z_j and Z_{j-1}. For all $j \leq J$ with $j > 1$ it holds that

$$\Pr(Z_j \geq T_i \mid Z_{j-1} \geq T_i) = \frac{\Pr(Z_j \geq T_i \cap Z_{j-1} \geq T_i)}{\Pr(Z_{j-1} \geq T_i)} = \frac{\Pr(Z_j \geq T_i)}{\Pr(Z_{j-1} \geq T_i)}. \tag{5.51}$$

It directly follows that

$$\Pr(Z_j \geq T_i) = \Pr(Z_j \geq T_i \mid Z_{j-1} \geq T_i) \Pr(Z_{j-1} \geq T_i) = \Pr(Z_1 \geq T_i) \prod_{o=2}^{j} \Pr(Z_o \geq T_i \mid Z_{o-1} \geq T_i). \tag{5.52}$$

Now suppose that there exists a bound q such that $\Pr(Z_j \geq T_i | Z_{j-1} \geq T_i) \leq q$ for all $j \leq J$ with $j > 1$. We can then also provide a bound for the error Err_z by rewriting equation (5.50) using the conditional probabilities:

$$\begin{aligned}
Err_z &= \sum_{j=1}^{J} \Pr\left(Z_j \geq T_i\right) \\
&= \Pr(Z_1 \geq T_i) + \Pr(Z_1 \geq T_i) \sum_{j=2}^{J} \prod_{o=2}^{j} \Pr(Z_o \geq T_i \mid Z_{o-1} \geq T_i) \\
&\leq \Pr(Z_1 \geq T_i) q^0 + \Pr(Z_1 \geq T_i) \sum_{j=2}^{J} q^{j-1} \\
&= \Pr(Z_1 \geq T_i) \left[\sum_{j=1}^{J} q^{j-1} \right] \\
&= \Pr(Z_1 \geq T_i) \left[\frac{1-q^J}{1-q} \right] \\
&\leq \Pr(Z_1 \geq T_i) \left[\frac{1}{1-q} \right]
\end{aligned} \tag{5.53}$$

For example, if the dependent probability is bounded by $q=0.05$ we get $Err_z \leq 0.053$ as $\Pr(Z_1 \geq T_i) \leq 0.05$ by definition. For $q=0.10$ we get $Err_z \leq 0.056$ and the error Err_z is still acceptable as it is not far off from our original significance level 0.05.

The bound $q=0.05$ has an intuitive interpretation as it relates to an independence assumption: if two random variables G and H are independent and satisfy $\Pr(G \geq T_i) \leq 0.05$ and $\Pr(H \geq T_i) \leq 0.05$ it follows immediately that

$$\Pr(G \geq T_i \mid H \geq T_i) = \frac{\Pr(G \geq T_i \cap H \geq T_i)}{\Pr(H \geq T_i)} = \frac{\Pr(G \geq T_i) \Pr(H \geq T_i)}{\Pr(H \geq T_i)} = \Pr(G \geq T_i) \leq 0.05 \tag{5.54}$$

and $q=0.05$ is a bound for $\Pr(G \geq T_i | H \geq T_i)$. Obviously, Z_1 and Z_2 are not independent, but we may still hope that $\Pr(Z_j \geq T_i | Z_{j-1} \geq T_i)$ is bounded.

We only study $\Pr(Z_2 \geq T_i | Z_1 \geq T_i)$ in the following, as the probability of falsely detecting a third recombination event is usually not as interesting anymore: first, $\Pr(Z_3 \geq T_i | Z_2 \geq T_i)$ may be similar to $\Pr(Z_2 \geq T_i | Z_1 \geq T_i)$. Second, if $\Pr(Z_2 \geq T_i | Z_1 \geq T_i) \leq 0.05$, we know that the probability of falsely detecting three recombination events is smaller than $0.05^2 = 0.0025$. Finally, the invariant presented in the next section suggests that it is extremely rare that more than two recombination events are falsely detected.

5.6.4.3.3 Invariants for the Savings Values

Obviously, a large $Savings_1$ also allows for a large $Savings_2$, as the condition $Savings_1 > Savings_2$ must always be satisfied. This condition suggests that the $Savings_k$ values as well as the z_j values are positively correlated and that $\Pr(Z_2 \geq T_i | Z_1 \geq T_i)$ is large. However, there is a more intricate restriction for $Savings_k$ and z_j that suggests a negative correlation between Z_1 and Z_2 for the relevant domain. If $Savings_1$ is particularly large, $Savings_2$ cannot be very large, because the recombination(s) associated with $Savings_1$ already reduce the number of mutations on the α-optimal path. The same restriction also applies to z_j. To be more specific, it follows from $Savings_k(R_{k+1}-R_k)=M_k-M_{k+1}$ that

$$\sigma = M_1 - M_K = \sum_{k=1}^{K-1}(M_k - M_{k+1}) = \sum_{k=1}^{K-1} Savings_k\,(R_{k+1} - R_k) = \sum_{j=1}^{J} z_j \ . \tag{5.55}$$

The interesting aspect about this equation is that σ does not change for column permutations of the alignment. The sum of z_j and $Savings_k$ therefore is the same for every column permutation of the alignment. If a column permutation causes a particularly large z_1 or $Savings_1$ value, the z_2 or $Savings_2$ value is probably much smaller than for other column permutations, as the total sum remains constant. If $J>1$ we can also bound z_2 by z_1:

$$\sigma - z_1 \geq \sum_{j=2}^{J} z_j \ . \tag{5.56}$$

In other words, if $z_1 > \sigma/2$ it must also hold that $\sigma/2 > z_2$ and z_2 has to decrease as z_1 increases. The initial restriction $z_1 \geq z_2$ becomes obsolete and both values are negatively correlated. As only large z_1 values can lead to a significant p-value, we may hope that z_1 and z_2 are non-correlated or negatively correlated for the relevant domain and that $\Pr(Z_2 \geq T_i | Z_1 \geq T_i)$ is small and can be bounded. These observations also carry over to the $Savings_k$ values.

5.6.4.3.4 Empirical Analysis

We first study the joint distribution of Z_1 and Z_2 and then come back to Y_1 and Y_2. Figure 5.24 shows an analysis of the joint distribution of Z_1 and Z_2 for the example alignment based on 100,000 column permutations. The mass of the distribution is not directly visible from the plot, as the distribution is discrete and each point may correspond to more than one sample. The red lines visualize T_i. A permutation with two falsely detected recombination events therefore corresponds to a point to the right of and above the red lines. The blue line visualizes an analogous threshold for Z_2, where the threshold t is the 95% quantile of Z_2 and satisfies $\Pr(Z_2 \geq t) \leq 0.05$. The blue curve represents t with $\Pr(Z_2 \geq t | Z_1 = z_1) \leq 0.05$, or the 95% quantile of Z_2 conditioned on Z_1. As expected, the blue curve often indicates a negative correlation between Z_1 and Z_2 if Z_1 is significantly large (to the right of the red line). The probability of detecting a second recombination event is less than 5% if the first recombination event is significant, as the blue curve is always below the dotted, red line. Figure 5.25 shows the same plot for the z_j values for the alignment. Table 5.2 shows several empirical probabilities for the example alignment, based

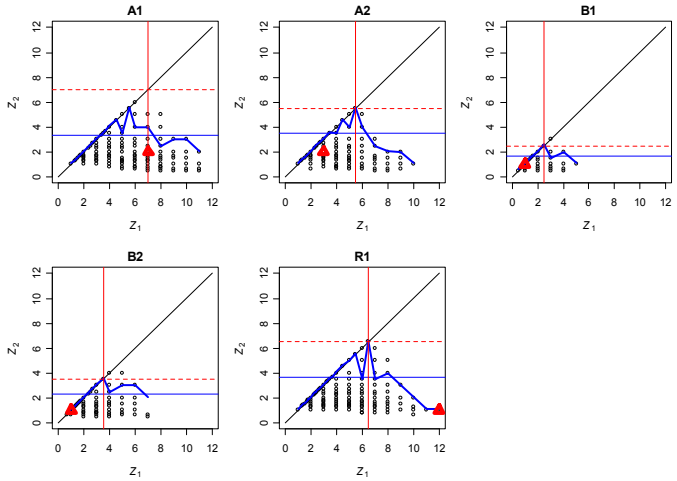


Figure 5.24: The scatterplots visualize the dependency between z_1 and z_2 for column permutations of the example alignment. Each plot corresponds to a different query sequence. The diagonals represent the constraint $z_1 \geq z_2$. The red line visualizes the threshold T_i and is also shown on the y-axis, as Recco uses T_i for all recombination events. The blue line shows the 95% quantile for Z_2 and the blue curve shows the 95% quantile for Z_2 conditioned on Z_1. The blue lines are not used by Recco, but visualize the dependency of Z_1 and Z_2 nicely. Finally, the red triangle marks the observed values z_1 and z_2 for the original alignment.

on 100,000 permutations. Recco is very conservative for the example alignment and only rarely detects a second recombination event if a first recombination event is detected.

The same analysis can also be conducted for $Savings_1$, $Savings_2$ and Y_1, Y_2. The negative correlation between Y_1 and Y_2 is more pronounced and the probability of falsely detecting a second Savings value is even lower than for Z_1 and Z_2 (not shown).

The analysis was repeated for two random sequence alignments of length N=1000 to ensure that the results also hold in other scenarios. The simulation process was the same for both alignments. First, a random genealogy was drawn from the coalescent using the program ms (Hudson 2002). Given the genealogy, we then used the program dawg (Cartwright 2005) to evolve a random sequence along the genealogy according to the Jukes-Cantor model. The first alignment was simulated with the parameters θ=500 and ρ=0 and therefore does not contain recombination. Hence, there were θ/N=0.5 mutations per position from the root to a leaf of the tree, on average (see section 3.3.2). The second alignment was simulated with the same scaled mutation rate θ=500, but used ρ=16 to introduce a substantial amount of recombination. However, the analysis only compares the distributions Z_1 and Z_2 derived from column permutations of the alignment and does not

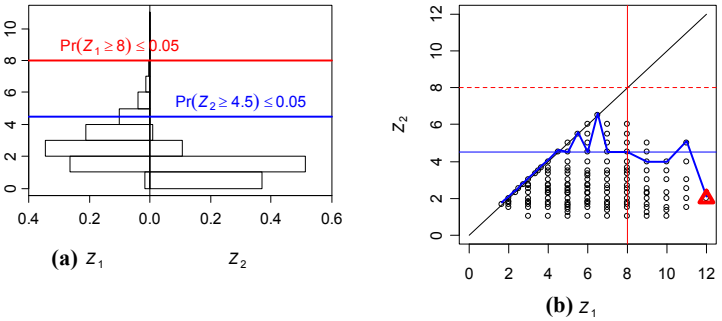


Figure 5.25: The data of Figure 5.24 were aggregated to the maximum z_1 and z_2 value over all sequences. **(a)** The histograms for Z_1 and Z_2 are plotted back to back. The threshold T for Z_1 is shown with a red line and is much higher than the 5% threshold for Z_2 shown with a blue line. Hence, the probability to detect a second recombination event with the threshold T is much lower than 5%. **(b)** The scatterplot shows the alignment values of Z_1 and Z_2 and confirms the negative correlation as before.

example alignment	alignment	R1	A1	A2	B1	B2
T, T_i	8	6.5	7	5.5	2.5	3.5
$\Pr(Z_1 \geq T_i)$	0.019	0.023	0.037	0.022	0.024	0.044
$\Pr(Z_2 \geq T_i)$	0.000	0.000	0.000	0.000	0.001	0.001
$\Pr(Z_2 \geq T_i \mid Z_1 \geq T_i)$	0.000	0.000	0.000	0.000	0.024	0.019

Table 5.2: The table lists the thresholds T and T_i for the alignment and the five sequences, respectively. The next rows list the probability to detect at least one recombination event $\Pr(Z_1 \geq T_i)$, the probability to detect a second recombination event $\Pr(Z_2 \geq T_i)$ and the dependent probability $\Pr(Z_2 \geq T_i \mid Z_1 \geq T_i)$. Note that $\Pr(Z_2 \geq T_i \mid Z_1 \geq T_i) = \Pr(Z_2 \geq T_i) / \Pr(Z_1 \geq T_i)$ in our case. By construction of the threshold T_i, $\Pr(Z_1 \geq T_i)$ and $\Pr(Z_2 \geq T_i)$ are always smaller or equal to 5%. The dependent probability is also well below 5% for the example alignment and shows that the probability to detect a second recombination event erroneously is lower than 5%.

study the probability of falsely detecting one or two recombination events beside the true recombination events. The alignment with ρ=16 guarantees that some permutations may achieve a very large Savings value. Finally, both alignments contain much more mutations than the example alignment and also have a much higher expected Savings value for column permutations.

θ=500, ρ=0	aln	1	2	3	4	5	6	7	8	9	10
T_i	23.5	1	2	2.33	2.33	3.5	5.5	6.5	11.5	11.5	23.5
$\Pr(Z_1 \geq T_i)$	0.041	0.046	0	0.022	0.042	0.012	0.020	0.041	0.041	0.046	0.041
$\Pr(Z_2 \geq T_i)$	0.000	0	0	0.014	0.018	0.002	0.001	0.000	0.000	0.000	0.000
$\Pr(Z_2 \geq T_i \mid Z_1 \geq T_i)$	0.001	0	0	0.629	0.441	0.166	0.057	0.001	0.001	0.000	0.001

θ=500, ρ=16	aln	1	2	3	4	5	6	7	8	9	10
T_i	27	1.25	1.66	2.66	4.33	4.33	4.66	5.5	8.66	13.5	27
$\Pr(Z_1 \geq T_i)$	0.046	0.022	0.029	0.026	0.034	0.047	0.049	0.024	0.049	0.038	0.046
$\Pr(Z_2 \geq T_i)$	0.000	0.017	0.009	0.005	0.015	0.019	0.014	0.003	0.005	0.000	0.000
$\Pr(Z_2 \geq T_i \mid Z_1 \geq T_i)$	0.000	0.768	0.310	0.172	0.446	0.407	0.278	0.125	0.095	0.006	0.000

Table 5.3: The same analysis as for Table 5.2 is shown for the two other alignments. The sequences are arbitrarily named 1 to 10, such that they are ordered by an increasing T_i. Probabilities above 5% are marked in red and indicate that the approach of Recco may detect many spurious recombination events if the threshold is particularly small.

θ=500, ρ=0	aln	1	2	3	4	5	6	7	8	9	10
T_i	23.5	1	2	2.33	2.33	3.5	5.5	6.5	11.5	11.5	23.5
$\Pr(Y_1 \geq T_i)$	0.041	0.046	0	0.022	0.042	0.012	0.020	0.041	0.041	0.046	0.041
$\Pr(Y_2 \geq T_i)$	0	0	0	0	0	0	0	0	0	0	0
$\Pr(Y_2 \geq T_i \mid Y_1 \geq T_i)$	0	0	0	0	0.004	0.001	0.004	0	0	0	0

θ=500, ρ=16	aln	1	2	3	4	5	6	7	8	9	10
T_i	27	1.25	1.66	2.66	4.33	4.33	4.66	5.5	8.66	13.5	27
$\Pr(Y_1 \geq T_i)$	0.046	0.022	0.029	0.026	0.034	0.047	0.049	0.024	0.049	0.038	0.046
$\Pr(Y_2 \geq T_i)$	0	0	0	0	0	0	0.001	0	0.001	0	0
$\Pr(Y_2 \geq T_i \mid Y_1 \geq T_i)$	0	0.003	0.001	0.002	0.008	0.008	0.010	0.001	0.011	0.001	0

Table 5.4: The same analysis as for Table 5.3 is shown, but for Y_1 and Y_2. Recco is extremely conservative in this analysis, as it usually does not falsely accept a second Savings value.

Table 5.3 lists the observed probabilities for the two alignments as described before for the example alignment. The threshold T is very conservative for both alignments and usually does not detect a second recombination event. However, the threshold T_i does not work well if T_i is small. For example sequence 3 in the alignment with ρ=0 corresponds to T_i=2.33. Recco falsely detects a second recombination event with a probability of 62.9% for this sequence if a first recombination event was detected. For larger T_i, the approach of Recco works well again and does not detect an excess amount of recombination events. Figure 5.26 shows the scatterplots for the Z_1 and Z_2 values for the alignment and illustrates that the threshold T is conservative. Finally, Figure 5.27 shows the sequence scatterplots for most sequences and confirms the problems discussed for Table 5.3. It is also very interesting to perform the same analysis for Y_1 and Y_2. Table 5.4 shows that Recco has a very low probability to falsely accept a second Savings value. In other

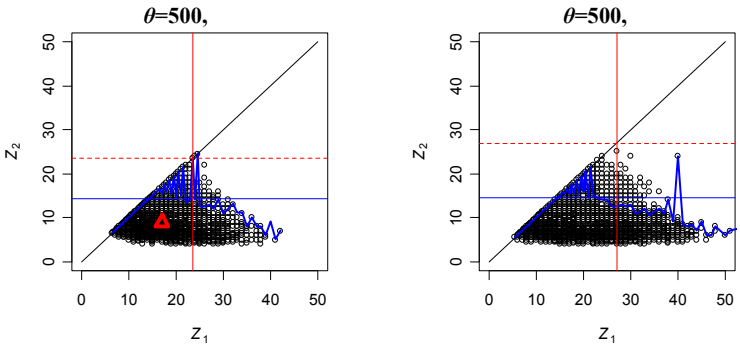


Figure 5.26: The scatterplots for the alignment values of Z_1 and Z_2 are shown for the two other alignments. The same trend is visible as before: significantly large Z_1 values usually lead to small Z_2 values. The red triangle of the right plot does not fall in the displayed range as it is far to the left and highly significant.

words, there is a simple reason why Recco may falsely detect more than one recombination event as in Table 5.3: the τ_i-optimal paths introduce more than one recombination event simultaneously.

In summary, the approach of Recco is very conservative and usually infers recombination events falsely in less than 5% of the cases. It is extremely rare that Recco chooses a threshold T_i such that two Savings values are falsely accepted. Consequently, Err_{Sav} is usually lower than 0.05 in practice. However, the threshold T_i may falsely detect more than one recombination event if T_i is small and Err_z may be large. This effect is usually not as relevant in practice, as these recombination events are introduced simultaneously into the τ_i-optimal paths. In other words, sequence i may not be a highly recombinant sequence if the corresponding T_i value is small and the τ_i-optimal paths recover many recombination events. A manual analysis could be performed to check if the α-optimal paths corresponding to the next larger Savings value are more likely in such a case. In practice, the thresholds T_i actually result in a more conservative estimate of the recombinant structure than other HIV-1 subtyping methods (see section 7.2.2). Finally, the threshold T is always conservative and even controls Err_z nicely.

5.6.5 Other Features for Recombination Detection

For notational convenience, the index for the query sequence i is dropped in the following and $Savings_1$ refers to the maximum Savings value of query sequence i. Besides the $Savings_1$ value, there are several other features of the cost curve that provide information on recombination in the query sequence. Many of these features do not perform as well as $Savings_1$ for recombination detection or are equivalent to $Savings_1$. One example is α_1, the smallest α-value for which a recombination is introduced in an α-optimal path. A

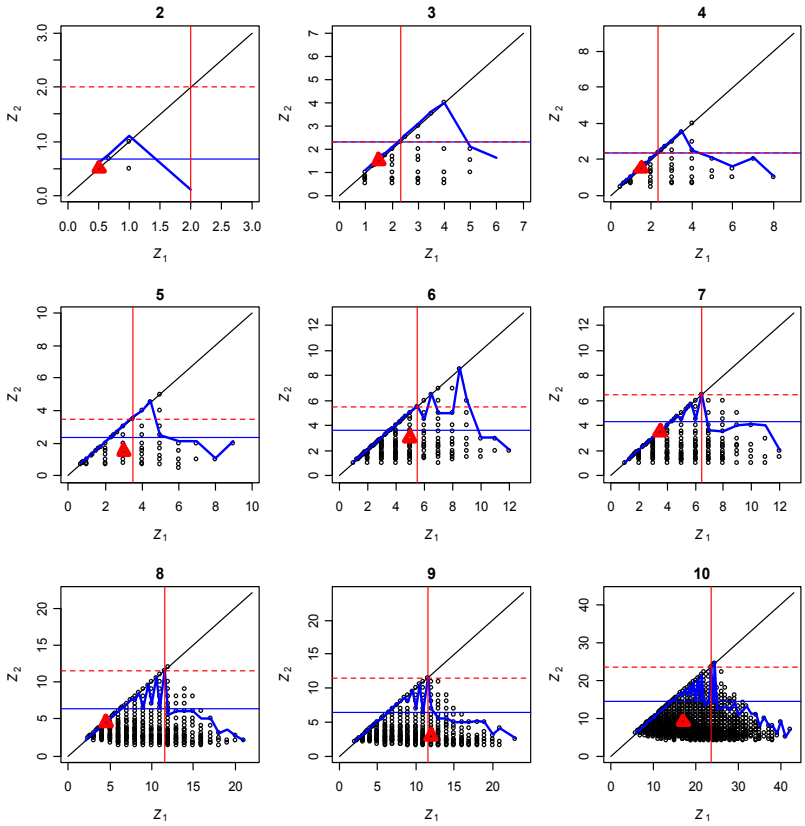


Figure 5.27: The scatterplots for the alignment with θ=500 and ρ=0 and almost all sequences. Sequence 1 is not interesting, as it does not erroneously detect a second recombination event. The problematic sequences are clearly visible from the plots. Sequence 3, 4, 5 and 6 show a positive correlation between Z_1 and Z_2 above the threshold T_i. Consequently, Recco often falsely detects more than one recombination event for these sequences simultaneously (see also Table 5.3).

p-value based on α_1 is equal to a *p*-value based on $Savings_1$: the relation $Savings_1=(1-\alpha_1)/\alpha_1$ is strictly monotonic for $\alpha_1 \in [0,1]$.

Another feature that is related but not equivalent to $Savings_1$ is *FirstAngle*, the angle in radians between the first and second linear segment of a cost curve (see Figure 5.28). A large angle indicates recombination as it points to a significant decrease of mutation cost and an increase of recombination cost. The last feature we propose is *MaxCost*, the maximum that the cost curve attains. This feature measures how hard it is to derive the putative recombinant from the other sequences in the alignment. The absolute value of

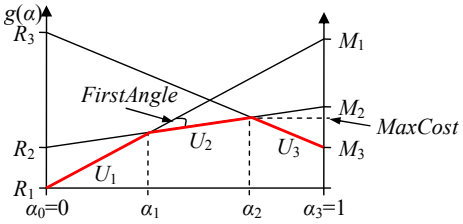


Figure 5.28: Several other features of the parametric cost curve can be used for detecting recombination in an alignment. Not all features are directly related to recombination, though.

MaxCost depends mostly on the time when the sequence diverged from the other sequences in the alignment. A sequence that diverged early in history tends to have a higher value of *MaxCost* than a sequence that diverged only recently. However, the absolute value of *MaxCost* does not matter for computing *p*-values. We have to compare *MaxCost* to the distribution expected under the null hypothesis. If the query sequence benefits much from introducing recombination, the slope of the cost curve decreases considerably at α_1 and *MaxCost* is rather small. The cost curve for a column permutation has the same slope for the first segment, but the slope does not decrease as much for the next segment, because recombination is not as beneficial. If the query sequence does not benefit from introducing recombination, we do not expect a significant change of *MaxCost*.
Both features, *FirstAngle* and *MaxCost*, can be used analogously to the Savings value for deriving *p*-values for recombination in the alignment or in a sequence. We could also determine the significance cutoff for *FirstAngle* and *MaxCost* as we did for the Savings value and recover the corresponding α-optimal paths. But the Savings value can also be computed for every segment in the cost curve, such that each recombination event corresponds to a Savings value. As this is impossible for *FirstAngle* and *MaxCost*, both features cannot be used to order the recombination event list or to compute *p*-values for a recombination breakpoint. *FirstAngle* and *MaxCost* also cannot be interpreted as easily and are not as intuitive as Savings, such that their application is limited. However, they capture a slightly different aspect of the recombination signal and are evaluated to some extent in Chapter 7.

5.6.6 Dealing with Similar Sequences in the Alignment

The basic approach of Recco seeks to recover the recent ancestry of the query sequence in the ARG. This approach imposes a fundamental restriction regarding recombination detection: Recco does not detect recombination in a query sequence if there is a closely related sequence in the alignment. The Savings value for R1 in the example alignment is small if R2 is included in the alignment. R2 effectively conceals the recombination signal of R1. This restriction of Recco is acceptable if we perform a manual analysis of the

alignment, as we can use the information on the recent ancestry of the query sequence to search for older recombination events. For example, we could reconstruct parts of the ARG incrementally, starting at the leaves and working towards the root of the ARG. This idea is quite involved, though, and some details are presented as future work.

A simpler approach is possible and can be performed automatically. If a query sequence does not benefit much from recombination, the query sequence is most likely involved in a coalescent event first. Hence, we can pick the closest sequence to the query and replace both sequences in the alignment by an intermediate sequence. The process is repeated until we find a significant recombination event. A slight variation is more suitable for practical purposes (see Figure 5.29). Starting with the full alignment, compute the $Savings_{1i}$ value for query sequence i and then eliminate the closest sequence to the query from the alignment. Repeat the process until there are only two sequences left in the alignment and record the maximum value of $Savings_{1i}$ encountered during this process. The maximum value of $Savings_{1i}$ is referred to as $OptSavings_i$ in the following. An alternative interpretation of the algorithm is that of a greedy strategy for finding the subset of sequences in the alignment that maximizes $Savings_{1i}$.

We can obtain p-values with the column permutation test again: compute $OptSavings_i$ for every column permutation and estimate the distribution of $OptSavings_i$ under the null hypothesis, such that the p-values for the observed values can be calculated. But $OptSavings_i$ introduces another limitation: $Savings_{ki}$ corresponds to the k-th transition in the parametric cost curve for query sequence i. The cost curve allows for recovering the recombination events introduced by each transition. But $OptSavings_i$ is the result of optimizing $Savings_{1i}$ and does not allow for reconstructing a recombination event list. As a workaround, we can evaluate the parametric cost curve that generated the $OptSavings_i$ value and refer to the Savings values of this cost curve as $OptSavings_{ki}$. Consequently, we can construct the recombination event list as before and assign OptSavings values to each recombination event. Hence, the same analysis pipeline as for $Savings_{ki}$ applies to $OptSavings_{ki}$. Unfortunately, it is difficult to interpret the recombination event list based on $OptSavings_{ki}$, as only the recombination events corresponding to $OptSavings_{1i}$ have an interpretation. The recombination events for $OptSavings_{1i}$ show the highest benefit (or Savings value) regarding a greedy optimization of the sequence subset. The other recombination events depend on the optimized sequence subset. It is possible, that the sequence subset conceals recombination events that would otherwise be part of the recombination event list. Hence, the interpretation of the recombination event list regarding the ARG is unclear if OptSavings is used. But for an automated analysis OptSavings performs very well.

Table 5.5 shows the recombination event lists based on the Savings value and the OptSavings value for the example alignment including sequence R2. OptSavings performs much better than Savings and recovers the recombination events that we expect for this scenario.

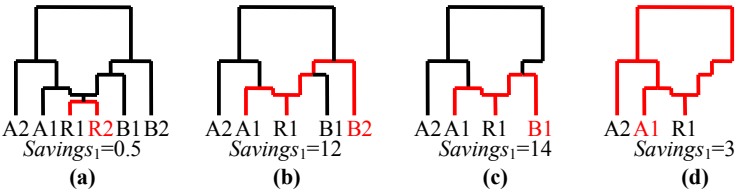


Figure 5.29: The iterations for computing the *OptSavings* feature for query sequence R1 in the example alignment are shown from left to right. The red sequence determines the sequence that is eliminated in the next step, as it is the closest sequence to R1. The red path visualizes the α-optimal path corresponding to the $Savings_1$ value. **(a)** R2 is closely related to R1 and does not allow for a high $Savings_1$ value. **(b)** After eliminating R2, we obtain the example alignment analyzed before. Interestingly, B2 is eliminated next, as it is closer to R1 than A1. **(c)** This sequence subset results in the maximum $Savings_1$ value. There is actually no better sequence subset for the example alignment. **(d)** As there are only three sequences left, it does not make sense to eliminate any further sequences.

i **(query)**	***j*** **(source)**	***j'*** **(dest)**	***p*** **(breakpoint)**	**Savings**	**Alignment *p*-value**	**Sequence *p*-value**
A1	R1	A2	45-46, 49-52	9	0.0075	0.0072
A2	B2	A1	19-24	3	0.9768	0.4898
…	…	…	…	…	…	…

i **(query)**	***j*** **(source)**	***j'*** **(dest)**	***p*** **(breakpoint)**	**OptSavings**	**Alignment *p*-value**	**Sequence *p*-value**
R2	A1	B1	50	15	0.0001	0.0001
R1	A1	B1	50	14	0.0001	0.0001
A1	R2	A2	45-46, 49-52	9	0.0335	0.0089
B2	A2	R1	24-26, 47-48	5	0.7562	0.0299
B1	A2	A1	58-74	4	0.9641	0.2844
A2	R2/B2	A1	19-24, 58-59	3	0.9995	0.4882
…	…	…	…	…	…	…

Table 5.5: The recombination event list is shown for the example alignment including sequence R2. *P*-values are based on 10,000 permutations, and all events with a Savings value greater or equal to three are shown. **(top)** The recombination event list based on Savings does not list sequence R1 and the associated recombination event. **(bottom)** The recombination event list based on OptSavings correctly lists the recombination events in sequences R1 and R2. The first three events in the list correspond to events expected from the ARG. All other events are not significant and constitute noise.

5.7 The Computational Complexity of Recco

Recco processes an alignment by building more and more complex analyses on top of each other. The steps in their hierarchical order are:

(a) The dynamic program (section 5.3). The dynamic program is very fast and only requires O(NM) operations for the forward, backward and backtrace steps if the heuristic of section 5.3.1 is used.

(b) The parametric analysis (section 5.4.2). The parametric analysis requires about $2K$ calls to the dynamic program if the cost curve has K segments. It is not simple to estimate K for a given alignment, as the number of segments depends on the diversity of the alignment and to a much lesser degree on the length of the alignment. K also varies for column permutations of the alignment. The example alignment has about 3.5 segments for each parametric cost curve, and this does not change much if the length is increased to N=1000. But more diverse alignments have ten or sometimes even one hundred segments in the cost curve.

(c) Computing the OptSavings values (optional). This step requires repeating (a) and (b) about M times, as the sequences are eliminated from the alignment.

(d) Taking each sequence in the alignment as a query. A full analysis of the alignment requires taking each sequence in the alignment as a query, in turn.

(e) The column permutation test (section 5.6.3). The column permutation test requires repeating steps (a)–(c) many times. If we consider p-values below 0.05 as significant, it is usually sufficient to perform 100 permutations.

Other parts of the analysis, such as constructing the recombination event list, do not require a lot of computation and are irrelevant for the running time of Recco. The approximate running time for steps (a) to (c) is usually less than one second for an alignment of N=1,000 positions and M=10 sequences. The total running time for an alignment of the same size is about 30 seconds if we do not compute $OptSavings_i$ and about 100 seconds if $OptSavings_i$ is also computed. The overall complexity of Recco is therefore O(NM^2KP) if $OptSavings_i$ is not computed and O(NM^3KP) otherwise. It would be more intuitive to express K in terms of M and N, as K does not directly depend on the input. An exact expression is not easily possible, but K grows slowly with the diversity and length of the alignment and the complexity of Recco is therefore slightly worse than linear in N.
There are several approaches that could increase the speed of Recco significantly. The permutation test is a particularly promising step for optimization. For example, the Savings value of column permutations appears to be Poisson-distributed (see Figure 5.21). If a theoretical analysis provides evidence for this assumption, we could simply estimate the parameter of the distribution and obtain accurate p-values from a few permutations. It

might also be possible to approximate the parameter of the distribution from simple summary statistics of the original alignment and avoid column permutations altogether.
Another approach reduces the running time for each column permutation and only requires a slightly more complex implementation. If we are not interested in *p*-values for recombination breakpoints, we can restrict the parametric analysis for the permuted alignments. To be more specific, we may choose to compute *p*-values only for some Savings values in the recombination event list, for example for Savings values greater or equal to 5. Consequently, it is sufficient to construct the cost curve for the interval [0,1/6], as α-optimal paths for $\alpha>1/6$ correspond to a Savings value smaller than 5 and are not relevant for computing *p*-values of the selected Savings values. The probability for a Savings value smaller than 5 is about 1% for the example alignment. Step (b) would therefore often require only a single call to the dynamic program instead of O(*K*) calls. However, some parts of the analysis of Recco cannot be performed in this case. The *MaxCost* feature, for example, requires most of the cost curve and could not be explored otherwise.

5.8 Conclusion

This chapter introduced Recco, a new method for the analysis of recombination in sequence alignments. The basic optimization problem and the dynamic program of Recco is very similar to several other approaches that use dynamic programming and cost minimization. All these approaches only differ by the set of allowed operations and by the input type. Recco was originally motivated by the work of Kececioglu and Gusfield (Kececioglu and Gusfield 1998) (see section 4.3.8.1), which reconstructs a single sequence from two other sequences using mutation, recombination and indels, and does not require a multiple alignment as input. Lajoie and El-Mabrouk (Lajoie and El-Mabrouk 2005) (see section 4.3.8.1) reconstruct one sequence in an alignment from the others using mutation, recombination and gene conversion. Jumping alignments (Spang, Rehmsmeier et al. 2000; Spang, Rehmsmeier et al. 2002) (see section 4.3.3.3) reconstruct one sequence from a multiple alignment of background sequences and allow for mutation, recombination and indels. The dynamic program of Recco is a restricted version of the last two approaches, as it does not allow for gene conversion or indels.
The dynamic program of Recco and the restrictions it imposes were carefully tailored to the requirements of recombination analysis in practice. The dynamic program of Recco allows for an intuitive visualization of *all* optimal solutions simultaneously and can also assess the robustness of the solution regarding small deviations from the optimal solution using the r_{ip} values. The other approaches could only visualize one optimal solution, and require more than two dimensions for a similar analysis of the robustness of a solution. The dynamic program of Recco is also considerably faster than the other approaches. Finally, Recco does not impose unrealistic assumptions, even though it is more restricted. Gene conversions can be represented as two independent recombination events. This may

reduce the accuracy of the model, but simplifies the dynamic program significantly. Jumping alignments can simultaneously align and analyze a query sequence regarding a reference alignment of sequences. But for general datasets it is usually unclear which sequence is the recombinant and each sequence of the dataset is considered as the query sequence in turn. In this case, there is virtually no benefit in automatically aligning the query sequence, as there is a different reference alignment for each possible query sequence.

Recco introduces several analyses that are not shared by other methods. The parametric analysis is the most informative and recovers all pareto-optimal paths. But the parametric analysis also allows for comparing neighboring segments of the parametric cost curve and assigns a Savings value to the transitions. The Savings value is an intuitive and quantitative measure for recombination in the alignment and in a sequence. The Savings value also provides a natural ordering for the recombination events in an α-optimal path, the recombination event list. The recombination event list is particularly useful for a manual analysis of the alignment, as it provides a condensed view of the recombination signal and allows focusing our attention to important events first. In some cases, the ordering might fail, though. The ARG Analysis explains how the Savings value and the α-optimal paths depend on the ARG underlying the alignment and encourages using the largest α in practice that does not introduce noise into the α-optimal paths. This also motivates interpreting α as the choice between robustness and resolution regarding recombination detours. The ARG Analysis also shows that Recco takes an entirely different approach to recombination analysis than many other methods: Recco does not try to infer local trees, but looks at the ARG bottom-up instead of left-right. This approach is particularly attractive, as recent events can be estimated far more accurately than older events in the presence of recombination. Finally, the column permutation test allows for computing p-values and helps to estimate the α for recovering robust α–optimal paths. As a consequence, Recco does not introduce any critical parameters into the analysis: we can usually use the value for α that is suggested by the column permutation test, or alternatively scan the recombination event list manually and pick a cutoff by hand.

Methods that are based on a similar optimization problem as Recco are often not adapted to the analysis of recombination and do not provide a connection between the optimization problem and inferring parts of an ARG. Even though they could benefit from the parametric analysis, the output would be difficult to interpret. There are also usually more than two types of cost for these methods and therefore more than one parameter in the parametric analysis. Methods testing the distribution of mutations, such as Geneconv and the MC^2 method, are designed to detect a recombination signal and do not introduce any parameters. Unfortunately, these methods also do not allow for an easy interpretation of the output regarding the ARG, and it is difficult to assemble a recombination hypothesis manually. Methods based on compatibility, such as PHI and NSS, only test for recombination and do not allow any further interpretation. Methods that compare local trees and

use a sliding window, such as the popular Bootscanning method, introduce a critical parameter: the sliding window size. The interpretation of the output is clear if the sliding window never covers a recombination breakpoint and is large enough to reconstruct the genealogy robustly. But this is usually not the case, as the window moves across the sequence. Methods that optimize the breakpoints and compare local trees, such as GARD, the MCPM or the HMM approach, overcome some of these problems and may correctly estimate trees for different regions of the alignment. But the interpretation of the sequence of local trees is difficult and does not directly correspond to the ARG, as recombination is not modeled as an SPR operation. Finally, the MARG method models recombination accurately and has a clear interpretation regarding the ARG – but only works for the infinite sites model and wrongly infers recombination in the presence of recurrent mutations. RecPars is probably a better compromise in this case. But the output of RecPars depends critically on the tradeoff between mutation and recombination cost. This tradeoff is not modeled and analyzed with a parameter like α.

Despite the merits of Recco, there is still ample room for improvement. Recco does not estimate the ARG and can only infer recent mutation and recombination events. The OptSavings value provides a simple workaround for automated recombination detection, but it is not as easy to interpret as the Savings value if the result is analyzed manually. Recco also does not correct distances for recurrent mutations, as it is based on parsimony. The connection to the ARG is only intuitive and simple if the estimated distances in Recco correspond to branch lengths in the ARG and becomes inaccurate as more and more recurrent mutations occur. Finally, handling gaps is a particular problem for Recco and is discussed in more detail in the next chapter.

6 Treating Gaps in Recco

Nucleotide sequences of the same gene across different species or individuals differ widely in length. A multiple sequence alignment of these sequences introduces gap characters into the sequences with the goal that all sequences have the same length and nucleotides at the same sequence position are similar to each other. Similarity between nucleotides is usually measured on an evolutionary, structural or functional scale and depends on the purpose of the multiple sequence alignment (Edgar and Batzoglou 2006). Hence, gap characters are a result of the multiple sequence alignment and have to be analyzed with great caution. While nucleotide bases in the same column of a multiple alignment represent homologous bases, this is not the case for gap characters. Two gap characters in the same column may represent two different insertion or deletion events and then do not represent similarity. It might be possible to partially resolve this ambiguity by inferring and encoding an evolutionary pathway that gave rise to the gaps (Simmons and Ochoterena 2000). Still, it is essential to realize that a gap character only indicates a missing homologous base and not an evolutionary event or a fifth nucleotide state. In the following, we refer to a single gap character as an indel and to a maximal consecutive stretch of indels as a gap.

Multiple sequence alignments are often the basis of subsequent analyses that build on the homology given by the alignment. For example, programs for inferring phylogenies use homology to estimate the evolutionary distance between different taxa. Recco uses homology to estimate cost (or distance) between the putative recombinant and the other sequences at each position. It is important that these methods can process gapped alignments correctly and do not return wrong or biased results. However, the specific strategy for treating gaps depends heavily on the problem itself and can vary greatly. A multiple alignment program may implement an affine linear gap cost function to penalize long gaps more than short gaps in total. For phylogenetic inference it is more appropriate to model insertion and deletion as evolutionary events or treat gaps as missing information. Another simple, but common strategy is to discard all columns that contain indels.

In the following, we discuss different strategies for handling gaps, evaluate their suitability for Recco and their potential impact on the output. Section 6.1 introduces several common strategies for dealing with gaps in phylogenetics and then discusses their application for recombination analysis. Treating gaps as missing data is particularly favorable but is not easily possible for Recco and several other methods. Section 6.2 presents how gaps enter the model of Recco, derives properties of reasonable gap cost functions and then introduces a general functional form for gap costs in Recco. A simple case is implemented in Recco and treats gap characters as a fifth nucleotide. Section 6.3 derives bounds for reasonable gap costs in Recco and proposes gap costs that share some characteristics with treating gaps as missing data. This analysis may eventually lead to a

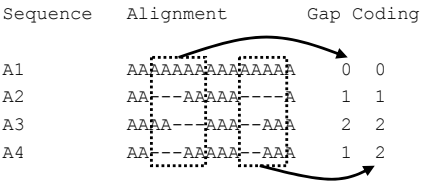


Figure 6.1: An example alignment that has two additional columns for coding gaps. Each region that contains overlapping gaps is coded independently with a separate character. Sequence A2 and A4 have the same gap in the first region and therefore are assigned the same character for coding the gaps of the first region.

strategy for treating gaps as missing data in Recco. A more detailed analysis and an implementation of the proposed gap costs are left for future research.

6.1 Related Work

A comprehensive summary and comparison of gap treatment in recombination detection methods would be very interesting. Unfortunately, many recombination detection methods ignore the impact that the treatment of gaps has on the analysis and do not explain the strategy that they implement. Particularly recombination detection methods that are based on a local reconstruction and comparison of phylogenetic trees often do not provide any details on their strategy, for example TOPAL, PLATO and Simplot (see chapter 4 for a description of these methods). However, they usually employ the strategies that are common for phylogenetic tree inference. In addition, many of these strategies are also applied in other recombination detection methods.

6.1.1 Strategies for Treating Gaps in Phylogenetic Inference

There are many strategies for treating gaps in phylogenetic analysis (see for example (Simmons and Ochoterena 2000; Ogden and Rosenberg 2006; Ogden and Rosenberg 2006)):

a) **Remove all columns from the alignment that contain an indel.** This strategy has the advantage that the analysis is unbiased, at least if the columns containing indels do not show special properties relevant for the analysis. It is also very simple to implement. Furthermore, gap regions in the alignment are often hard to align, thus removing them from the analysis may actually remove noise. Unfortunately, removing gap columns from the alignment also discards a lot of information and potentially leads to a higher variance of the estimated output.

b) **Treat indels as a fifth nucleotide state.** This option is mainly used for inferring phylogenetic trees by maximum parsimony or distance methods and incorporates gap information. But the underlying assumption that each indel is an independent event is

usually wrong. It can only be justified if a gap should involve a cost linear in its length.

c) **Treat indels as missing data.** The resulting effect is best explained for the problem of inferring phylogenetic trees by maximum parsimony. Before computing the mutation cost for a fixed tree topology, indels are replaced by the best fitting nucleotide for the tree. In other words, indels are effectively treated like a wildcard and gaps are filled with sequence data from the closest node in the tree. Maximum parsimony then chooses the tree topology that results in the smallest mutation cost, optimizing the wildcard for each topology separately. The result is that indels do not count towards the cost of the phylogenetic tree and thus do not influence the tree. Even though this is a rather conservative choice, it discards all information on gaps. For example, if two sequences share a gap and a third does not, but the sequences are equal otherwise, they are all regarded as equal by this strategy.

d) **Code gaps as separate characters.** A newer strategy in phylogenetics codes gaps as a separate character in the alignment (see Figure 6.1) and uses the information on gaps for inferring the tree (Simmons and Ochoterena 2000; Cheynier, Kils-Hütten et al. 2001; Young and Healy 2003). Each gap state in the alignment is encoded as a different character and the transition cost between these characters approximates the cost of the insertion and deletion events that are required for a transition between the corresponding gap states. A standard maximum parsimony tree for the augmented alignment then minimizes the total cost of mutation, insertion and deletion events along the tree and approximates a cost minimal history of these events. However, gap coding only codes gaps and gap states that are present in the alignment. The parsimonious tree therefore cannot infer intermediate gap states that are not present in the alignment. Furthermore, the methods penalize each transition between the different gap states equally and do not implement a model that incurs a larger penalty for larger gaps.

e) **Infer the history of a set of sequences allowing for mutation, insertion and deletion events.** We can extend the concept of parsimony and infer the history of a set of sequences minimizing the cost of mutation, insertion and deletion events (Schwikowski 1998). In comparison to d), such a strategy allows for reconstructing intermediate gap states that are not contained in the alignment. The same conceptual strategy is also possible in a maximum likelihood framework (Lunter, Miklós et al. 2003).

The first three strategies are widespread strategies in phylogenetic analysis, d) is uncommon and was only proposed for maximum parsimony and e) requires a lot of computational power and is still subject to research.
Ogden and Rosenberg (Ogden and Rosenberg 2006) studied the accuracy of maximum parsimony tree inference on simulated sequence alignments and compare the strategies

b), c) and d). Treating gaps as missing data was inferior to the other strategies if the simulated sequence alignments were directly used as an input. In this case, the alignment is accurate and treating gaps as missing data does not use the information on gaps for estimating the ancestry more accurately. To study the effect of errors in the alignment, the authors realigned the sequences using ClustalW and therefore generated alignments that are closer to our expectation in practice. All three methods achieved a similar accuracy on this dataset, probably because the information on gaps was too unreliable to be useful. In conclusion, it is not possible to identify the single best strategy for real sequence data, at least if all nucleotides and encoded gaps are weighted equally as in the article. Consistent results across different strategies can increase the reliability of the inference, though.

6.1.2 Strategies for Treating Gaps in Recombination Analysis

Most of the strategies used in phylogenetics directly apply to methods for recombination analysis. But phylogenetic tree inference differs from recombination analysis regarding the assumptions about the input and the desired output. Phylogenetic tree inference often assumes for the input that each site evolves independently from each other, whereas recombination analysis depends on the ordering of the sites in the alignment and additionally has to estimate sequence similarity for local regions in the alignment. The resulting complications are manifold. For example, treating indels as missing data in phylogenetic tree inference imputes the indels of a sequence using the nucleotides of the closest or best fitting sequence globally. But we cannot refer to the best fitting sequence globally for recombination analysis anymore as we have to evaluate sequence similarity locally. The column permutation test can also become inappropriate. It basically tests whether the particular ordering of columns in the input alignment is significant – but gaps consist of many consecutive indels and also depend on the ordering of columns. A column permutation distributes the indels of a gap across the alignment. The permuted alignment is therefore structurally different from the original alignment and potentially leads to wrong *p*-values and to falsely detected recombination events. The specific effect depends on the strategy for treating gaps, though. Finally, the output of recombination analysis has to estimate the breakpoint of a recombination event and therefore critically depends on the spatial resolution of the breakpoint hypotheses. Recombination analysis can become inaccurate or even biased if we do not allow to infer a recombination breakpoint between every position of the alignment. These differences regarding the input and the output of the analysis make it much more difficult to treat gaps appropriately for recombination analysis when compared to phylogenetic inference.

Our goal for recombination analysis is usually to recover as many true recombination events as possible while keeping the probability of falsely detecting recombination events at a minimum. The reasoning is that a falsely detected recombination event often has much stronger implications. It may wrongly indicate that the sequences evolved subject to recombination and therefore suggest a totally different mode of evolution for the or-

ganism. This asymmetry does not prevail for all tasks in recombination analysis, though. Our conclusions may not depend as much on a falsely detected recombination event if we want to estimate the recombinant structure of a sequence and know that there is a fair amount of recombination in the alignment. In this case, only some part of the estimate is wrong and the overall conclusion is not affected as severely. In any case, we can usually control the false detection rate nicely with the column permutation test if there are no gaps in the alignment (see chapter 7). As described above, gaps in the alignment can interfere with the column permutation test and make it much more difficult to control the false detection rate. In summary, there are several desirable properties for recombination analysis that are difficult to achieve:

i) **Never infer a recombination event based on indels only.** As described above, we are interested in detecting and reconstructing only true recombination events and therefore have to control the number of falsely detected recombination events. This is difficult as most strategies for treating gaps may falsely infer recombination events only because of a gap. Consequently, we only want to infer a recombination event if the evidence comes from polymorphisms other than indels and would rather want to miss the recombination event otherwise. In other words, we do not want to use the information on gaps for inferring recombination events. This objective is very important but rather imprecise and is discussed in more detail in section 6.2.2.3.

ii) **It is appropriate to use the column permutation test.** The column permutation test is an elegant approach to assessing the significance of a recombination event. However, it depends on the strategy for treating gaps whether the test correctly controls the false detection rate for recombination events. The reason is simple: each gap may cover hundreds of nucleotides and an indel therefore does not represent an independent event. If we permute the columns of the alignment, we distribute the indels of each gap across the alignment. Consequently, permuted alignments often differ significantly from non-permuted alignments only because of the shorter gap lengths. Some strategies for treating gaps result in wrong and possibly significant *p*-values for recombination based on this effect.

iii) **The information on polymorphisms in gap regions is retained.** A strategy for treating gaps could simply discard all columns containing an indel. But estimating the recombinant structure of a single sequence may not depend on all other sequences and in particular may not depend on sequences containing a gap. For example, the red and blue paths in Figure 6.2 do not touch and do not depend on the gap. Hence, it is desirable to keep all polymorphisms in gap regions for the analysis such that we can estimate the recombinant structure more accurately.

		a)	b)	c)	d)	e)
i)	never infer a recombination event based on indels only	-	-	+	-	(-)
ii)	the column permutation test is applicable	+	(-)	?	+	-
iii)	information on polymorphisms in gap regions is retained	-	+	+	+	+
iv)	can infer recombination breakpoints in gap regions	-	+	+	-	+
v)	number of additional parameters for gaps	0	1	0	≥1	≥1

Table 6.1: The strategies for treating gaps are compared regarding several properties. All properties are formulated such that a '+' indicates a desirable state. Values in brackets indicate that the property depends on the implementation of the strategy and are usually discussed in more detail in the text describing each strategy.

iv) Recombination breakpoints in gap regions can be inferred. Some strategies for treating gaps do not allow for inferring recombination events inside gap regions and can limit the spatial resolution of recombination analysis considerably. For example, if we encode a gap region as a separate character, we cannot infer a recombination breakpoint within the gap region as this would split the character that encodes the gap region into two pieces. As a result, the accuracy of the breakpoint estimate suffers. Even worse, the overall estimate can also become affected adversely.

v) The number of parameters introduced for treating gaps should be limited. A high number of parameters is not by itself negative. However, it is often difficult to set the parameters correctly if the strategy for treating gaps introduces many parameters and if the output of the recombination analysis depends critically on them.

Some of these properties are interrelated. For example, we can i) falsely infer recombination events if we iii) discard polymorphisms in gap regions or iv) disallow recombination in gap regions (see Figure 6.2). We can now study in more detail if it is easy to transfer the strategies for treating gaps to recombination analysis and if the strategies have an adverse impact on the properties described above (see Table 6.1):

a) Remove all columns from the alignment that contain an indel. As described before, this approach is particularly simple to implement, but discards a lot of information. We cannot infer recombination events within gap regions and may therefore loose some accuracy or miss some recombination events. Even worse, we discard all information on polymorphisms in gap regions even if the gaps do not interfere with the analysis of the query sequence under study. Although the loss of information is a rather subtle effect, it may cause wrong results under special circumstances and eventually lead to falsely detected recombination events (see Figure 6.2).

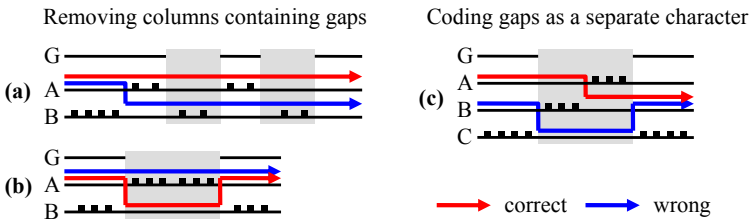


Figure 6.2: Strategies for treating gaps can fail in special situations. Black lines represent sequences in the alignment. A space in the line and a grey background indicate a gap. Sequence G always contains one or more gaps, but is not relevant for the analysis otherwise. The goal is to estimate the recombinant structure of the query sequence. The query sequence is not shown in the alignment, but the black dots on the sequences represent mutations with respect to the query sequence. The red arrow specifies the true recombinant structure and the blue arrow shows a wrong estimate if gaps are treated inappropriately. Even though the visualization of the estimates particularly resembles Recco's approach, other methods for recombination analysis produce similar results. **(a)** Sequence A fits to the query sequence well. The few mutations are not sufficient to support recombination events if gap regions are included in the analysis. However, if we remove the gap regions from the alignment, we also remove all mutations in sequence B and infer a recombination event erroneously. **(b)** We may also miss a recombination event if we remove all gap regions from the alignment. Missing a recombination event is usually not as critically as falsely detecting a recombination event and the first probably also occurs more frequently than the latter in practice. **(c)** Coding gaps as a separate character can also cause inferring false recombination events. The recombination event of the red path is supported by mutations in the gap region. But the recombination event cannot be inferred if we do not allow for recombination events inside the gap region. Consequently, the blue path predicts two false recombination events.

b) **Treat indels as a fifth nucleotide state.** This strategy retains all polymorphisms in gap regions and even uses the information on gaps for the estimate. However, such a strategy is very sensitive to the cost assigned for a transition to or from an indel state. Each gap can cover tens or hundreds of nucleotides and introduces polymorphisms of the same length. Depending on the cost setting, a gap may be detected as a significant clustering of polymorphisms and lead to falsely detected recombination events. The column permutation test is also not as easily applicable anymore, as a large gap causes a sequence of polymorphisms that are scattered across the alignment in column permutations. As a result, recombination events often do not only depend on the cost parameter for scoring indels, but the *p*-value of the recombination event also changes

in a complex way as the cost parameter is adjusted. This strategy and the impact it has on the output of Recco is discussed in more detail in section 6.2.3.

c) **Treat indels as missing data.** Treating indels as missing data is a conservative choice for phylogenetic tree inference, as indels then do not influence the inferred tree. But it depends on the particular problem how we should treat or impute missing data appropriately and it is not easy to transfer the idea from phylogenetic tree inference to recombination analysis in general. As described before, we only want to infer a recombination event if the evidence comes from polymorphisms other than indels only and we have to impute missing data accordingly. Nevertheless, this objective is quite imprecise and several specific strategies for different recombination analysis methods are discussed in section 6.1.2.1. The properties listed in Table 6.1 are therefore only a rough guideline.

d) **Code gaps as separate characters.** Coding gaps also retains all polymorphisms in gap regions and uses the information on gaps for the estimate, just like strategy b). In comparison to this strategy, coding gaps does not allow for inferring recombination events in the gap region. It is difficult to find an appropriate cost for scoring gaps or indels and as a consequence we may detect false recombination events (see Figure 6.2).

e) **Infer the history of a set of sequences allowing for mutation, recombination, insertion and deletion events.** This strategy is computationally infeasible even if we do not allow for indels. We also have to decide on the cost for insertion and deletion in addition to the cost for mutation and recombination. However, the estimate could be more accurate than the output for any other strategy if the parameters are set appropriately.

Apparently, there is no single best strategy for treating gaps in recombination analysis as any method fails for some situations. Treating gaps as missing data is probably the most attractive strategy as it avoids inferring recombination events based on gap information only and also compares favorably with respect to the other strategies (see Table 6.1).

6.1.2.1 Treating Gaps as Missing Data

Treating gaps as missing data is favorable compared to other strategies, but the implementation of such a strategy inherently depends on the method for recombination analysis. In the following, we discuss several methods that allow for treating gaps as missing data quite easily. The discussion often does not refer to the actual strategy that is implemented in a method for recombination analysis, as this information is frequently unpublished. A notable exception is the program RDP (Martin, Williamson et al. 2005) (http://darwin.uvigo.es/rdp/rdp.html) that implements several methods for recombination analysis and also discusses how it treats gaps.

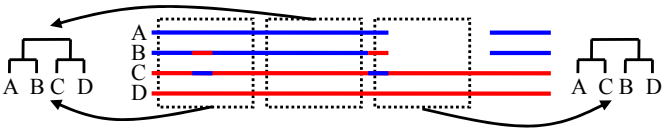


Figure 6.3: A recombination analysis method that estimates local trees has to use a measure of robustness, as it can otherwise falsely detect recombination events based on very few polymorphisms. Shown is an alignment of four sequences, where each line corresponds to a sequence and colors indicate homology. A missing line represents a gap in the sequence and a dotted box indicates a sliding window. The small strips of homology that conflict with the overall homology could be caused by random mutations. The first and the second sliding window support the same tree topology, because the conflicting signal in the first window is not strong enough. The third sliding window covers the gap region and does not have much sequence information for sequences A and B. The tree estimated from the third window conflicts with the other trees, and we can detect a recombination event if we do not take into account the robustness of the trees. But the third tree is only supported by very few polymorphisms and is not robust. Taking this into account, the trees may not differ significantly anymore.

Methods that infer a phylogenetic tree on a sliding window or a local coalescent tree during their computation often transfer the idea from phylogenetic tree inference directly. Particularly sliding window-based methods (for example Bootscanning, PLATO and TOPAL) usually assume that some appropriate strategy for treating gaps is used during phylogenetic tree inference. Treating gaps as missing data often works well for these methods. But gaps may cause problems even in this case, because the partitions of a tree inferred from a gapped region are sometimes based only on a few polymorphisms (see Figure 6.3) or because two overlapping sliding-windows impute the gap differently. It is therefore essential that we also compute a measure of reliability for each tree partition, for example by bootstrapping (see section 4.3.4.1). The bootstrap values can then be used to assess whether two neighboring trees differ significantly and often avoid falsely inferring recombination events at gap boundaries. Bootscanning (see section 4.3.4.1) uses such a strategy, but apparently does not process the information appropriately and still detects spurious recombination events for specific alignments (see Figure 6.4). A similar strategy is possible for other methods that infer a local tree as well. For example, methods that find the optimal position for recombination breakpoints and infer local trees between the breakpoints (see section 4.3.5 and 4.3.6) can also treat gaps as missing data as in phylogenetic inference. Some of these methods use the same basic strategy within a Bayesian framework for inference and can detect significant changes of local coalescent trees directly from the posterior without bootstrapping.

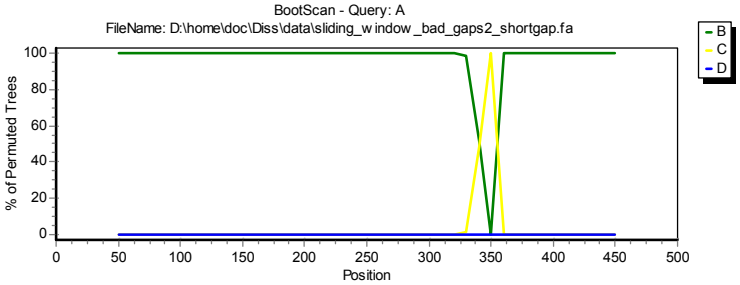


Figure 6.4: The plot shows the output of Simplot (see section 4.3.4.1) using a window size of 100 bases, a step size of ten bases and sequence A as a query for the bootscanning plot. The input alignment is schematically visualized in Figure 6.3. Strips of conflicting homology are ten bases long and occur at positions 101-110 and 301-310. We either expect to detect no recombination at all or to explain both strips of conflicting homology by two recombination events. Even though bootscanning uses a measure of robustness, it does not treat the gap region correctly and detects a recombination event only for the second strip of conflicting homology. The alignment is very challenging in general and several other methods either crash (PDM and several implementations in RDP) or also detect a recombination event only at the gap boundary (TOPAL). GARD is an exception and detects as expected two recombination events around both strips of conflicting homology.

Interestingly, the fundamental building block of this strategy for treating gaps is not phylogenetic inference. It is only important that each sliding window or each region between the proposed recombination breakpoints does not contain a recombination event. In this case, we can impute the missing data for each gap by phylogenetic inference or alternatively by using the most similar sequence as a template. As before, we just have to assess potential recombination events very carefully as we may impute the gap based on very few polymorphisms or impute it differently for adjacent sliding windows.

Methods based on compatibility (see section 4.3.2) can implement a particularly elegant strategy for treating gaps as missing data, as the concept of compatibility easily extends to gaps. Indels simply do not cause incompatibility as we can safely assume that each indel represents a compatible nucleotide.

Finally, the maximum chi-squared method (see section 4.3.1.1) can also treat gaps as missing data, as it is based on processing pairs of sequences only. We can simply remove all positions with an indel in either of the two sequences for each sequence pair that we process (see the RDP Manual at http://darwin.uvigo.es/rdp/rdp.html for more details). This only results in a minimal loss of information and effectively treats indels as missing data.

6.1.2.2 Other Strategies for Treating Gaps Implemented in Practice

As described above, many methods for recombination analysis do not provide any information on the implemented strategy for treating gaps. Most methods based on a sliding window or on estimating local trees treat gaps as missing data by default, for example Bootscanning, PhylPro and GARD. RecPars computes the maximum parsimony cost of a column given a tree topology and can also treat gaps as missing data. The implementation of TOPAL in TOPALi (Milne, Wright et al. 2004) recently introduced an undocumented "gap threshold" setting, resulting in less false positives due to gaps than previous versions of TOPALi. In 2007 and before, TOPALi apparently treated a gap as a fifth nucleotide and weighted it equally to other polymorphisms. Gap regions often caused different trees inside and outside the gap region, for example when two very similar sequences do not share a gap. A large gap almost always resulted in a strong recombination signal at the first and last gap position. Finally, compatibility methods and the maximum chi-squared method use the strategies outlined before.

Several methods and papers do not treat gaps as missing data and propose a different strategy. The authors of some phylogeny-based methods such as PDM discard columns containing indels before the analysis (Etherington, Dicks et al. 2005; Husmeier, Wright et al. 2005) to avoid potential problems with treating gaps. Geneconv determines runs of contiguous sites containing indels that are maximal in length and treats them as single, large polymorphisms. This gap coding strategy could accommodate for arbitrarily complex transition cost between the large polymorphisms. But Geneconv assigns the same weight to each polymorphism even if it covers a large gap region. Large gap regions then contribute only very little to the overall diversity and the effect is very similar to that of removing gap regions. The HIV-1 subtyping method jpHMM implements a very interesting strategy, as it also models insertion and deletion events explicitly in the hidden Markov model. But jpHMM requires a large set of training sequences to estimate the probability of insertion and deletion events as well as the transition probabilities between nucleotides. A large training set is not a problem for HIV-1 subtyping, but it is much more difficult in the general case. It would be necessary to tune the parameters manually or estimate them from sequences other than the sequences under study.

6.2 Gaps Entering the Model of Recco

Most strategies for treating gaps that were described before can also be implemented for Recco and have similar consequences for the analysis. We can easily implement the strategies a) remove all columns containing an indel, b) treat indels as a fifth character and d) code gaps as separate characters. But Recco does not use a fixed size sliding window and therefore we cannot easily c) treat gaps as missing data. Strategy a) is particularly simple and does not require a separate analysis. Strategy d) seriously limits the possible locations of recombination breakpoints just like a). Hence, our focus in this section is on treating gaps as a fifth nucleotide and on treating gaps as missing data.

As a first step, we extend the cost function of Recco and add a term for scoring gaps. We are very flexible scoring gaps in Recco from a technical point of view, as we only require that the gap cost function is computable by dynamic programming. But not all gap cost functions are relevant in practice. Using simple arguments we can motivate several desirable properties of the gap cost function and can restrict the set of potential cost functions considerably. We then proceed studying the types of artifacts in the output of Recco in more detail and obtain a better understanding for the complexity of treating gaps appropriately. A first result of this analysis is a cost function that treats gaps as a fifth nucleotide if they occur in the α-optimal path.

6.2.1 A Modified Cost Function

We can extend the cost function that was introduced in Chapter 5 and add a term for scoring gaps. The cost of a path is then composed of the total cost for recombination, mutation and gaps. The cost for recombination and mutation differ qualitatively: the cost for recombination only depends on the path **u**, while the cost for mutation is computed by comparing the putative recombinant sequence **s** and the sequence **s'** that is implied by **u**, but it does not directly depend on the path **u**. Clearly, the cost for gaps should only depend on the two sequences **s** and **s'**, just like the cost for mutation. Thus, the decision about a gap cost function is the decision about a function taking as input two sequences of equal length, **s** and **s'**.

Now define a gap indicator $g_A(i,p)$ with $g_A(i,p)=1$ if A_{ip} is an indel and $g_A(i,p)=0$ otherwise and let $g_{\mathbf{s}}(p)$ be the gap indicator for the putative recombinant sequence **s**. Define $G_\alpha(\mathbf{u},x,y)$ as the cost function that scores gaps in path **u**, where G_α implicitly depends on the gap indicator functions. The only computational requirement for G_α is that it can be computed by dynamic programming. We clarify the structure of G_α in the following. In analogy to (5.3), the total cost of path **u** between positions x and y is

$$\varphi_\alpha(\mathbf{u},x,y) = (1-\alpha)R(\mathbf{u},x,y) + \alpha M(\mathbf{u},x,y) + G_\alpha(\mathbf{u},x,y)\,. \tag{6.1}$$

As we still want to compute the cost function φ_α by dynamic programming, we assume that we can decompose φ_α as

$$\varphi_\alpha(\mathbf{u},x,y) = \varphi_\alpha(\mathbf{u},x,z) + \varphi_\alpha(\mathbf{u},z+1,y) + \tau_\alpha(u_z,u_{z+1},z) \tag{6.2}$$

for any x,y,z with $x \le z < y$ where $\tau_\alpha(u_z,u_{z+1},z)$ is some transition cost. The transition cost models the cost for recombination, but can also incorporate some costs for scoring gaps. The general form of the cost function above motivates the following form for G_α

$$G_\alpha(\mathbf{u},x,y) = \sum_{p=x}^{y-1} \gamma_\alpha(u_p,u_{p+1},p) + \sum_{p=x}^{y} \eta_\alpha(u_p,p) \tag{6.3}$$

where γ_α and η_α are arbitrary cost functions that could, for example, resemble recombination and mutation cost, respectively.

The terms γ_α and η_α are very flexible for designing gap cost functions. We can model arbitrary gap extension cost or substitution costs for indels with $\eta_\alpha(u_p,p)$ and therefore we assume without loss of generality that $m(i,p)=0$ for all i and p with $g_A(i,p)=1$ or $g_{\mathbf{s}}(i,p)=1$.

The term $\gamma_\alpha(u_p,u_{p+1},p)$ can actually model a fixed cost for opening a gap in either of the two sequences **s** and **s'**, as γ_α depends on the nucleotides and indels that path **u** covers at position p and p+1.

It is also important that G_α depends on α. Mutations in the α-optimal path incur a high cost if α is low, but their cost decreases as α approaches 1. If G_α assigns a fixed cost to a gap that does not depend on α, the α-optimal path may incorporate the gap for α close to 0, but has to avoid the gap as α increases because mutations are cheaper relative to the gap in this case. In other words, the parameter α would determine whether we incorporate a gap into the α-optimal path and therefore results in an undesirable effect. It is more appropriate to weight gaps by α similar to mutations such that α still controls the amount of recombination and does not force the α-optimal path to avoid gaps. Section 6.3.4 provides more details and shows that a good gap cost function should include terms weighted by α and terms weighted by $(1-\alpha)$.

6.2.2 A First Analysis

We can motivate several properties of the gap cost function without a deeper mathematical analysis and restrict the space of relevant gap cost functions.

6.2.2.1 Recco is Not Pairwise Alignment

The gap cost function compares a pair of sequences, **s** and **s'**. It is tempting to use a function similar to the gap costs for pairwise alignment and use gap open and gap extension cost for scoring gaps. But there are several aspects that we have to take into account. First, the two sequences are not the result of a pairwise alignment. Both sequences are constructed from a multiple sequence alignment and an indel in one sequence can be paired with an indel in the other sequence. This case does not indicate a real indel or a missing homologous base and should not be penalized. A gap cost function should therefore result in the same cost if all positions that pair two indels in **s** and **s'** are removed before evaluating the cost function. This is only a technical aspect and can be handled easily in the dynamic programming formulation. Hence, we limit our discussion in the following to cases where an indel in the recombinant is paired with a base in the explanation or vice versa.

Second and more importantly, the column permutation test does not work reliably anymore and may cause wrong results if we introduce a gap open cost. We have already described this effect before and provide a more detailed justification here. A column permutation rips apart each gap and distributes it across the alignment, giving rise to several smaller gaps. Clearly, the α-optimal paths of a column permutation usually do not incorporate a region covering multiple gaps if the gap open cost is high. As a consequence, the probability of a large Savings value in column permutations is greater if the gap open cost is high compared to situations without gap open cost. Hence, the original alignment dif-

fers significantly from column permutations with respect to costs and the p-values are not correct anymore.
Finally, scoring gaps with gap open and extension cost also requires that we set and tune two parameters appropriately. But it is even difficult to tune a single gap parameter as there is no real sequence data with a know history of recombination events.

6.2.2.2 Gap Costs in the Query Sequence and in the α-Optimal Path Differ

Another important insight is that an indel in the query sequence and an indel in the α-optimal path are qualitatively different. The asymmetry is founded in the goal of Recco: reconstruct the query sequence from the other sequences in the alignment with minimum cost. As a consequence, nucleotides in the α-optimal path that are paired with an indel in the query are superfluous and represent additional information as they are not necessary for explaining the query. In evolutionary terms this might be the result of a non-homologous recombination or a deletion event that removes part of a sequence. An appropriate cost penalty for such an indel would be small or even zero. On the other hand, indels in the α-optimal path imply missing information as some parts of the query sequence were not derived from the given sequence alignment. Thus, the cost for indels in the α-optimal path should be higher than for indels in the query sequence.
Now assume that there is a gap in the query sequence. If the gap incurs zero cost, that means pairing an indel to another nucleotide is not penalized, the α-optimal path is indifferent about the choice of the explanatory sequence at the gaps' positions. The effect is the same as removing all columns from the alignment where the query sequence contains an indel thus discarding much less information than removing all columns that contain an indel in any sequence. As the cost for the gap increases, the α-optimal path prefers pairing indels in the query with indels in the path and may introduce additional recombination events. These recombination events are only caused by the attempt of pairing indels with each other (see Figure 6.5). But the paired indels do represent homology and may not even correspond to the same evolutionary insertion or deletion event. In other words, the resulting recombination events are not based on reliable information. Therefore, we assume in the following that indels in the query sequence incur a cost of zero without regard to the indel or nucleotide they are paired with and set $\eta_\alpha(i,p)=0$ for all i and p where $g_s(p)=1$. This strategy is conservative and resembles treating gaps as missing data. As an equivalent strategy, we can also remove all columns with an indel in the query sequence and then simply assume that there are no indels in the query sequence for all subsequent analysis steps. Hence, we only have to handle the case when indels in the α-optimal path are paired with nucleotides in the query sequence in the following.

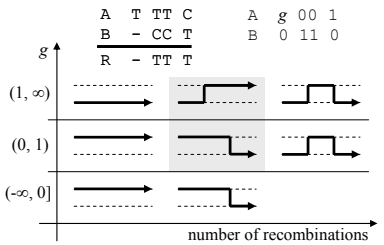
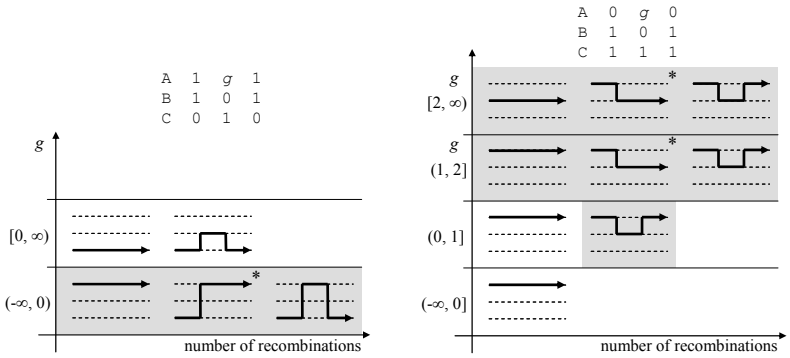


Figure 6.5: The plot describes the α-optimal paths for an alignment containing an indel in the query sequence R. Only the mutation cost matrix at the top right is relevant for the α-optimal paths. The alignment at the top left is displayed for illustrative purposes only. Each α-optimal path is represented in the plot by an arrow through the cost matrix. Each row shows the α-optimal path for an increasing parameter α, i.e. an increasing number of recombination events. The ordering is qualitative only and columns do not necessarily represent similar α-values. The cost of pairing an indel with a nucleotide is also kept variable and is represented by g on the y-axis and in the cost matrix. The bottom row in the plot corresponds to gap costs below zero that are too low to make sense and are only included for completeness. Particularly interesting for the given scenario are the paths highlighted by a grey background. They show that a gap can force the α-optimal path through a specific sequence, which may then affect the path in regions further away from the gap.

6.2.2.3 The Impact of Gaps on the α-Optimal Path

This section analyzes the impact of a gap on the α-optimal path if the gap occurs in a sequence other than the query sequence. In essence, we assume that an indel is treated as a fifth nucleotide and then increase the cost of an indel from zero to infinity, observing changes in the α-optimal path. We also assume that the cost of the indel is weighted multiplicatively by α just like the mutation cost, so that α still controls the amount of recombination in the α-optimal path correctly. The following analysis is general enough to represent many other scenarios despite our focus on a single indel in the alignment, as a gap that covers several consecutive columns often results in the same output as a combined column.

Figure 6.6 shows two cases where it is clear whether the gap should be part of the α-optimal path. Even though these cases are too simple for real world scenarios they illustrate important artifacts in the α-optimal path. The gap may involve less cost than any other alternative sequence or path if we use a very low gap penalty. As a result, the α-optimal path is forced to cover the gap and may introduce recombination events solely for this purpose (see Figure 6.6 a), bottom row). In an extreme scenario, the explanation



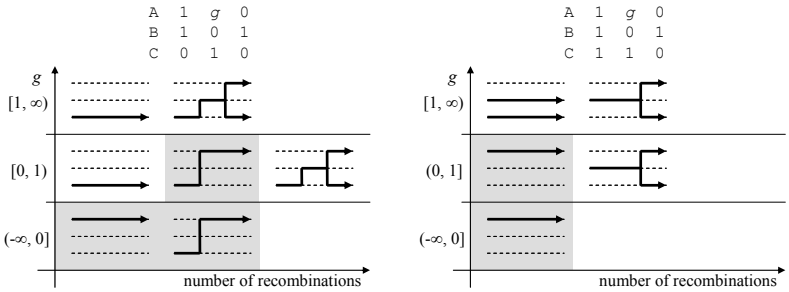
(a) no gap use **(b)** gap use almost certain

***** there exists a vertically symmetric alternative for the shown α-optimal path.

Figure 6.6: Each plot describes the α-optimal paths for an alignment as in Figure 6.5. In comparison to Figure 6.5 the gap occurs in the alignment and not in the query sequence and a grey background highlights paths that do not make sense. The alignments visualize very simple cases where the desired output of Recco is clear. Only extreme settings for the cost of an indel cause a different output. **(a)** Sequence B is identical to A in the regions excluding the gap and involves the least possible cost in the gap region. Consequently, an α-optimal path should always prefer sequence B over sequence A. **(b)** Sequence A fits the recombinant perfectly, but has a gap. The α-optimal path covers sequence A even if we replace the gap with the worst alternative cost and score it with a cost of 1.

might even jump to a gapped sequence only to avoid a region of higher mutation cost. Obviously, these recombination events do not make sense. As the gap cost increases, the gap loses its advantage over the other sequences. Eventually, the gap incurs such a high cost that it is avoided by the α-optimal path under all circumstances. Even if the optimal solution mainly uses the sequence which contains the gap, the gap itself is not part of the solution anymore and additional recombination events are introduced into the path (see Figure 6.6 b), top row). Figure 6.7 presents two alignments that are more difficult to interpret and do not show such strong artifacts.

In summary, we have to choose the cost associated with an indel or a gap carefully, as gaps should not induce spurious recombination events due to a high or low gap cost. This is impossible to achieve for some alignments if we penalize all indels in the alignment equally. For example, indels have to be penalized heavily if other sequences in the gap region fit poorly to the query sequence as the indels would otherwise attract the α-optimal path and introduce spurious recombination events. But a second gap may not require such a high gap penalty, for example because the other sequences fit the query well. Even



(a) gap use unclear **(b)** gap use unclear

Figure 6.7: The plots visualize the α-optimal paths for more complex alignments where the desired output is not clear. As in Figure 6.6, a grey background highlights paths that do not make sense. **(a)** Overall, sequence A fits the query better than B, but not as well as C. The mutation cost in the gap region of sequence A lies between zero and one. Such a cost penalty also seems plausible for penalizing the gap region and recovers the α-optimal paths in the middle row of the plot. But it is not clear if the path in the center of the plot is supported by the data. The path introduces a recombination event only to avoid the mutation cost in sequence C at least if the gap in sequence A incurs a cost $g<1$. The gap incurs a lower penalty than the mutation in sequence C in this case. **(b)** This scenario is less problematic as all reasonable gap costs lead to the same recombination breakpoints. Nevertheless, it is unclear whether the paths with a grey background are reasonable as sequence A is predicted as the most similar sequence to the query sequence overall.

worse, a high indel cost can force the α-optimal path to avoid the gap by all means, introducing additional recombination events. Hence, the α-optimal path can predict recombination events to include or exclude a gap for such an alignment no matter how we set the cost of an indel. We could avoid such an artifact by penalizing indels differently depending on their position in the alignment. Section 6.3 provides more details on such a strategy.

6.2.2.4 A Simple Gap Cost Function

We can now consolidate our insights regarding the gap cost function and define the functional form in more details. In summary, the gap cost function should

- **a)** not have a term for gap open cost.
- **b)** not penalize gaps in the query sequence.
- **c)** not cause spurious recombination events due to a high or low gap cost.

The last point is too vague for restricting the gap cost function and is the basis for a more detailed analysis in section 6.3. We also assume that the term γ_α is always zero even though γ_α does not necessarily represent gap open cost. But the analysis in the following is much easier without this term and γ_α is also not necessary for a good gap cost function. In summary, we propose the following gap cost function:

$$G_\alpha(\mathbf{u}, x, y) = \sum_{p=x}^{y} \eta_\alpha(u_p, p) \tag{6.4}$$

where $\eta_\alpha(u_p,p)=0$ if $g_\mathbf{s}(p)=1$, so that we do not penalize a gap in the query sequence.

6.2.3 Treating Gaps as a Fifth Nucleotide

The analysis of HERV sequences in section 7.2.1 required that we implement a strategy for treating gaps. As we have outlined before, our options for treating gaps in Recco are limited as we cannot easily treat gaps as missing data. A practical alternative is to treat an indel as a fifth nucleotide and implement a strategy that helps detecting potential artifacts and questionable recombination events in the output. These recombination events can then be validated manually.

The basic idea of this strategy is to use the simple gap cost function of section 6.2.2.4 and penalize all indels in the α-optimal path with the same cost g, thereby setting $\eta_\alpha(i,p)=g$ for all i and p with $g_A(i,p)=1$ and $g_\mathbf{s}(p)=0$ and setting $\eta_\alpha(i,p)=0$ otherwise. We then vary g between 0 and 1 and perform a full Recco analysis for each setting. The output of all runs is consolidated automatically as described in section 7.2.1.2. In essence, recombination events that are only predicted for some settings of g are considered questionable and should be analyzed manually.

A particularly interesting outcome of the analysis in section 7.2.1 is a characterization of the ambiguous recombination events. These recombination events were not only caused by a gap that attracts or repels the α-optimal path, but also by more complex effects. For example, the Savings value of a recombination event can vary together with the gap penalty even if the α-optimal path does not cover the gap. Section 7.2.1 provides a classification of questionable recombination events together with several examples and explains the protocol of the analysis in more detail.

6.3 First Steps for Treating Gaps as Missing Data in Recco

A gap cost function should ideally treat gaps as missing data – or at least prevent the artifacts described before. But the gap cost function introduced in section 6.2.2.4 does not even prevent us from inferring recombination events only because of a gap. To be more specific, gap costs above or below certain bounds always exclude or include the gap in the α-optimal path. These artifacts can be avoided if the cost of a gap lies between certain bounds and we can derive even better bounds for the gap costs based on other simple arguments. Even though the following analysis is still incomplete and does not conclude

with a specific gap cost function, it provides a solid foundation for future research and may eventually allow for treating gaps as missing data in Recco.

6.3.1 Definitions and Prerequisites

To limit the complexity of the analysis, we assume that $r(i,j,p)=r(p)\delta(i,j)\geq 0$ such that the cost for a recombination does not depend on the donor and acceptor sequences i and j. We clearly indicate if we also require that $r(p)$=const for all p such that the cost of recombination does not vary across positions. We also assume that all positions with a gap in the query sequence are removed from the alignment before the analysis such that we can limit the analysis to gaps in the α-optimal path.

The derivation is based on decomposing the cost functions G_α and φ_α for any path $\mathbf{u}$ and any z with $x\leq z<y$:

$$\begin{aligned} G_\alpha(\mathbf{u},x,y) &= G_\alpha(\mathbf{u},x,z)+G_\alpha(\mathbf{u},z+1,y) \\ \varphi_\alpha(\mathbf{u},x,y) &= \varphi_\alpha(\mathbf{u},x,z)+\varphi_\alpha(\mathbf{u},z+1,y)+(1-\alpha)r(z)\delta(u_z,u_{z+1}) \end{aligned}. \tag{6.5}$$

Now define the set of paths that do not use an indel between positions x and y as

$$P(-,x,y)=\left\{\mathbf{v} \mid g_A(v_p,p)\neq 1 \text{ for all } p\in[x,y]\right\} \tag{6.6}$$

and the set of paths that do not touch path $\mathbf{u}$ anywhere between positions x and z as

$$P(\mathbf{u},x,y)=\left\{\mathbf{v} \mid v_x\neq u_x,\ldots,v_y\neq u_y\right\}. \tag{6.7}$$

We also refer to a path $\mathbf{v}\in P(\mathbf{u},x,y)$ as an alternative path to $\mathbf{u}$. We usually only refer to an alternative path if path $\mathbf{u}$ covers a gap between x and y and satisfies $g_A(u_p,p)=1$ for all $p\in[x,y]$. The bounds we derive are based on the cost of the best alternative path to $\mathbf{u}$ or the best path that does not use indels between positions x and y:

$$B_\alpha(-,x,y)=\min_{\mathbf{v}\in P(-,x,y)}\left\{\varphi_\alpha(\mathbf{v},x,y)\right\} \text{ and } B_\alpha(\mathbf{u},x,y)=\min_{\mathbf{v}\in P(\mathbf{u},x,y)}\left\{\varphi_\alpha(\mathbf{v},x,y)\right\}. \tag{6.8}$$

If path $\mathbf{u}$ covers a gap between x and y it follows that $B_\alpha(-,x,y)\geq B_\alpha(\mathbf{u},x,y)$ because $P(-,x,y)\subseteq P(\mathbf{u},x,y)$. Finally, we can also base gap costs on the path $\mathbf{u}^*$ that does not use any indels and incurs minimal cost:

$$\mathbf{u}^*=\arg\min_{\mathbf{v}\in P(-,1,N)}\left\{\varphi_\alpha(\mathbf{v},1,N)\right\}. \tag{6.9}$$

If there is more than one path $\mathbf{u}^*$, we break ties arbitrarily. Furthermore, we introduce a bound based on $\mathbf{u}^*$

$$B_\alpha^*(-,x,y)=\varphi_\alpha(\mathbf{u}^*,x,y). \tag{6.10}$$

6.3.2 Lower and Upper Bounds for Gap Cost in Recco

The following lower and upper bounds are based on a simple observation: if a gap involves a very low or a very high cost, it must be included or avoided by the α-optimal path irrespective of the sequence context (see Figure 6.6 a) bottom right and Figure 6.6 b) top right). The central observation is given by Lemma 1 and Lemma 2 then introduces the bounds for the gap cost function.

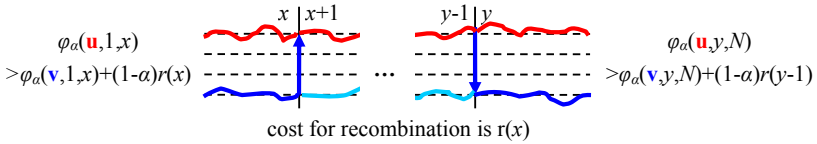


Figure 6.8: The plot illustrates the ideas of Lemma 1. Dotted lines represent the sequences in the alignment excluding the query sequence. The red and blue lines represent path **u** and **v** and are scribbled to indicate that the paths can have recombination events anywhere in the alignment. The middle part of path **v** is light blue as it is not relevant for the illustrated analysis. Path **u** is not (α,x+1)-prefix optimal if it incurs a higher cost than using path **v** between positions 1 and x plus a recombination event to path **u** at position x. The same observation follows for the end of the alignment independently.

Lemma 1: Let α be constant and consider an arbitrary path **u**. If there is some path **v** and some x with $\varphi_\alpha(\mathbf{u},1,x)-\varphi_\alpha(\mathbf{v},1,x)>(1-\alpha)r(x)$ then **u** is not (α,x+1)-prefix optimal. Similarly, if there is some path **v** and some y with $\varphi_\alpha(\mathbf{u},y,N)-\varphi_\alpha(\mathbf{v},y,N)>(1-\alpha)r(y-1)$ then **u** is not (α,y-1)-suffix optimal. Finally, if there is some path **v** and some x and y with $x\leq y$ and $\varphi_\alpha(\mathbf{u},x,y)-\varphi_\alpha(\mathbf{v},x,y)>(1-\alpha)(r(x-1)+r(y))$ then **u** is not (α,y+1)-prefix optimal and not (α,x-1)-suffix optimal. Path **u** is also not α-optimal in these cases as it is a necessary condition that an α-optimal path is prefix and suffix optimal.

Proof:

As defined in section 5.3 a path **u** is (α,x+1)-prefix optimal if and only if all paths **v** with $v_{x+1}=u_{x+1}$ satisfy $\varphi_\alpha(\mathbf{u},1,x+1)\leq\varphi_\alpha(\mathbf{v},1,x+1)$. Now suppose there is a path **v** and an x with $\varphi_\alpha(\mathbf{u},1,x)-\varphi_\alpha(\mathbf{v},1,x)>(1-\alpha)r(x)$. We can then show that the path $\mathbf{k}=(v_1,\ldots,v_x,u_{x+1},\ldots,u_N)$ contradicts with the prefix optimality of **u** as shown in Figure 6.8. Consider the decomposition of paths **u** and **k**:

$$\begin{aligned}\varphi_\alpha(\mathbf{u},1,x+1)&=\varphi_\alpha(\mathbf{u},1,x)+\varphi_\alpha(\mathbf{u},x+1,x+1)+(1-\alpha)r(x)\delta(u_x,u_{x+1})\\ \varphi_\alpha(\mathbf{k},1,x+1)&=\varphi_\alpha(\mathbf{v},1,x)+\varphi_\alpha(\mathbf{u},x+1,x+1)+(1-\alpha)r(x)\delta(v_x,u_{x+1})\end{aligned}.$$

It follows that

$$\begin{aligned}\varphi_\alpha(\mathbf{u},1,x+1)-\varphi_\alpha(\mathbf{k},1,x+1)&=\left[\varphi_\alpha(\mathbf{u},1,x)-\varphi_\alpha(\mathbf{v},1,x)\right]+(1-\alpha)r(x)\left[\delta(u_x,u_{x+1})-\delta(u_x,u_{x+1})\right]\\&>(1-\alpha)r(x)+(1-\alpha)r(x)\left[\delta(u_x,u_{x+1})-\delta(v_x,u_{x+1})\right]\\&\geq 0\end{aligned}$$

because $\delta(a,b)-\delta(c,d)\geq -1$ for all a,b,c,d. Thus, **u** is not (α,x+1)-prefix optimal because $k_{x+1}=u_{x+1}$ and $\varphi_\alpha(\mathbf{u},1,x+1)>\varphi_\alpha(\mathbf{k},1,x+1)$.

The other proofs are very similar, but are provided for completeness. Suppose there is a path **v** and some y with $\varphi_\alpha(\mathbf{u},y,N)-\varphi_\alpha(\mathbf{v},y,N)>(1-\alpha)r(y-1)$. Define a path $\mathbf{k}=(u_1,\ldots,u_{y-1},v_y,\ldots,v_N)$ and decompose paths **u** and **k**

$$\varphi_\alpha(\mathbf{u}, y-1, N) = \varphi_\alpha(\mathbf{u}, y-1, y-1) + \varphi_\alpha(\mathbf{u}, y, N) + (1-\alpha)r(y-1)\delta(u_{y-1}, u_y)$$
$$\varphi_\alpha(\mathbf{k}, y-1, N) = \varphi_\alpha(\mathbf{u}, y-1, y-1) + \varphi_\alpha(\mathbf{v}, y, N) + (1-\alpha)r(y-1)\delta(u_{y-1}, v_y)$$

It follows that

$$\begin{aligned}\varphi_a(\mathbf{u}, y-1, N) - \varphi_\alpha(\mathbf{k}, y-1, N) &= \left[\varphi_a(\mathbf{u}, y, N) - \varphi_\alpha(\mathbf{v}, y, N)\right] + (1-\alpha)r(y-1)\left[\delta(u_{y-1}, u_y) - \delta(u_{y-1}, v_y)\right] \\ &> (1-\alpha)r(y-1) + (1-\alpha)r(y-1)\left[\delta(u_{y-1}, u_y) - \delta(u_{y-1}, v_y)\right] \\ &\geq 0\end{aligned}$$

and therefore $\varphi_\alpha(\mathbf{u},y\text{-}1,N)>\varphi_\alpha(\mathbf{k},y\text{-}1,N)$ and $\mathbf{u}$ is not $(\alpha,y\text{-}1)$-suffix optimal as $k_{y-1}=u_{y-1}$.

Finally, assume there is a path $\mathbf{v}$ and some x and y such that $\varphi_\alpha(\mathbf{u},x,y)\text{-}\varphi_\alpha(\mathbf{v},x,y)>(1\text{-}\alpha)(r(x\text{-}1)+r(y))$. We can assume that $x>1$ and $y<N$, because otherwise the first part of the Lemma directly applies. Define a path $\mathbf{k}=(u_1,\ldots,u_{x-1},v_x,\ldots,v_y,u_{y+1},\ldots,u_N)$ and consider the decompositions

$$\varphi_\alpha(\mathbf{u}, 1, y) = \varphi_\alpha(\mathbf{u}, 1, x-1) + \varphi_\alpha(\mathbf{u}, x, y) + (1-\alpha)r(x-1)\delta(u_{x-1}, u_x)$$
$$\varphi_\alpha(\mathbf{k}, 1, y) = \varphi_\alpha(\mathbf{u}, 1, x-1) + \varphi_\alpha(\mathbf{v}, x, y) + (1-\alpha)r(x-1)\delta(u_{x-1}, v_x)$$

leading to the following inequality

$$\begin{aligned}\varphi_\alpha(\mathbf{u}, 1, y) - \varphi_\alpha(\mathbf{k}, 1, y) &= \left[\varphi_\alpha(\mathbf{u}, x, y) - \varphi_\alpha(\mathbf{v}, x, y)\right] + (1-\alpha)r(x-1)\left[\delta(u_{x-1}, u_x) - \delta(u_{x-1}, v_x)\right] \\ &> (1-\alpha)\left(r(x-1) + r(y)\right) + (1-\alpha)r(x-1)\left[\delta(u_{x-1}, u_x) - \delta(u_{x-1}, v_x)\right] \\ &\geq (1-\alpha)r(y)\end{aligned}.$$

Consequently, the first part of the Lemma applies and path $\mathbf{u}$ is not $(\alpha,y\text{+}1)$-prefix optimal as there is a path $\mathbf{k}$ and some y with $\varphi_\alpha(\mathbf{u},1,y)\text{-}\varphi_\alpha(\mathbf{k},1,y)>(1\text{-}\alpha)r(y)$. We can similarly conclude that path $\mathbf{u}$ is not $(\alpha,x\text{-}1)$-suffix optimal by decomposing $\varphi_\alpha(\mathbf{u},x,N)$ at position y

$$\varphi_\alpha(\mathbf{u}, x, N) = \varphi_\alpha(\mathbf{u}, x, y) + \varphi_\alpha(\mathbf{u}, y+1, N) + (1-\alpha)r(y)\delta(u_y, u_{y+1})$$
$$\varphi_\alpha(\mathbf{k}, x, N) = \varphi_\alpha(\mathbf{v}, x, y) + \varphi_\alpha(\mathbf{u}, y+1, N) + (1-\alpha)r(y)\delta(v_y, u_{y+1})$$

leading to

$$\begin{aligned}\varphi_\alpha(\mathbf{u}, x, N) - \varphi_\alpha(\mathbf{k}, x, N) &= \varphi_\alpha(\mathbf{u}, x, y) - \varphi_\alpha(\mathbf{v}, x, y) + (1-\alpha)r(y)\left[\delta(u_y, u_{y+1}) - \delta(v_y, u_{y+1})\right] \\ &> (1-\alpha)\left(r(x-1) + r(y)\right) + (1-\alpha)r(y)\left[\delta(u_y, u_{y+1}) - \delta(v_y, u_{y+1})\right] \\ &\geq (1-\alpha)r(x-1)\end{aligned}$$

and it follows from the first part of the Lemma that $\mathbf{u}$ is not $(\alpha,x\text{-}1)$-suffix optimal.

□

Lemma 2: Suppose there is a gap in sequence i covering positions a to b and a path $\mathbf{i}=(i,\ldots,i)$ that covers the gap. A reasonable gap cost function G_α must then satisfy for all x and y with $a\leq x\leq y\leq b$

$$B_\alpha(\mathbf{i}, x, y) - (1-\alpha)\left(r(x-1) + r(y)\right) \leq G_\alpha(\mathbf{i}, x, y) \leq B_\alpha(\mathbf{i}, x, y) + (1-\alpha)\left(r(x-1) + r(y)\right)$$

as an α-optimal path must otherwise either include or exclude part of the gap irrespective of the sequence context of the gap.

Proof:

Suppose that $G_\alpha(\mathbf{i},x,y)<B_\alpha(\mathbf{i},x,y)\text{-}(1\text{-}\alpha)(r(x\text{-}1)+r(y))$ for some x and y with $a\leq x\leq y\leq b$. We show that the gap cost function does not penalize gaps strong enough in this case. Con-

sider an arbitrary path $\mathbf{u}$ with $u_p \neq i$ for all $p \in [x,y]$ such that it avoids the gap between positions x and y. By definition, we know that $\mathbf{u} \in P(\mathbf{i},x,y)$ and therefore it follows that $B_\alpha(\mathbf{i},x,y) \leq \varphi_\alpha(\mathbf{u},x,y)$ as $B_\alpha(\mathbf{i},x,y)$ is the minimal cost of a path in $P(\mathbf{i},x,y)$. Remember that path $\mathbf{i}$ covers the gap from a to b and that $\varphi_\alpha(\mathbf{i},x,y) = G_\alpha(\mathbf{i},x,y)$ for any x and y with $a \leq x \leq y \leq b$. We obtain

$$\begin{aligned}\varphi_\alpha(\mathbf{i},x,y) &= G_\alpha(\mathbf{i},x,y)\\ &< B_\alpha(\mathbf{i},x,y) - (1-\alpha)\big(r(x-1)+r(y)\big)\\ &\leq \varphi_\alpha(\mathbf{u},x,y) - (1-\alpha)\big(r(x-1)+r(y)\big)\end{aligned}$$

and it follows from Lemma 1 that path $\mathbf{u}$ is not α-optimal. Any α-optimal path $\mathbf{u}$ therefore satisfies $u_p = i$ for some $p \in [x,y]$ and includes at least part of the gap. In other words, gap costs below the lower bound force an α-optimal path to cover some part of the gap.

Now suppose that $G_\alpha(\mathbf{i},x,y) > B_\alpha(\mathbf{i},x,y) + (1-\alpha)(r(x-1)+r(y))$ for some x and y with $a \leq x \leq y \leq b$ and consider an arbitrary path $\mathbf{u}$ with $u_p = i$ for all $p \in [x,y]$. By definition of $B_\alpha(\mathbf{i},x,y)$ there exists a path $\mathbf{v} \in P(\mathbf{i},x,y)$ with $v_p \neq i$ for all $p \in [x,y]$ and $\varphi_\alpha(\mathbf{v},x,y) = B_\alpha(\mathbf{i},x,y)$. We get

$$\begin{aligned}\varphi_\alpha(\mathbf{u},x,y) = G_\alpha(\mathbf{u},x,y) &= G_\alpha(\mathbf{i},x,y)\\ &> B_\alpha(\mathbf{i},x,y) + (1-\alpha)\big(r(x-1)+r(y)\big)\\ &= \varphi_\alpha(\mathbf{v},x,y) + (1-\alpha)\big(r(x-1)+r(y)\big)\end{aligned}$$

and $\mathbf{u}$ is not α-optimal due to Lemma 1. Any α-optimal path $\mathbf{u}$ therefore satisfies $u_p \neq i$ for some $p \in [x,y]$ and exclude at least part of the gap.

□

6.3.3 An Improved Lower Bound for Gap Costs

The bounds introduced in Lemma 2 are necessary to avoid artifacts in the α-optimal path, but they are not sufficient. We can actually improve the lower bound considerably using a simple argument: an α-optimal path should never introduce recombination events to incorporate only the gap or part of it. To be more specific, any α-optimal path that recombines into a gap and out of the gap without incorporating any surrounding nucleotide of the gapped sequence is non-sense. Based on this observation, we can derive an improved lower bound for the gap cost. Lemma 3 provides the essential insight and Lemma 4 obtains the result about gap costs.

Lemma 3: Let α be constant and consider an arbitrary path $\mathbf{u}$ with recombination events at the positions $\Delta(\mathbf{u}) = \{x | u_x \neq u_{x+1}\}$. Path $\mathbf{u}$ is α-optimal if and only if $\varphi_\alpha(\mathbf{u},x+1,y) \leq \varphi_\alpha(\mathbf{v},x+1,y)$ for all paths $\mathbf{v}$ and all $x,y \in \Delta(\mathbf{u}) \cup \{0,N\}$ with $x < y$. In other words, a path is α-optimal if and only if the cost of the path between any two recombination breakpoints is lower or equal to the cost of any other path between these breakpoints.

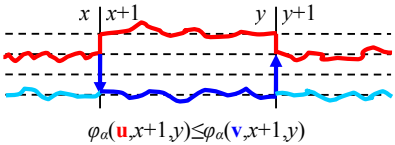


Figure 6.9: The plot illustrates the idea of Lemma 3. As in Figure 6.8, the dotted lines represent all sequences excluding the query sequence and scribbled lines indicate that the path could contain arbitrary recombination events. Clearly, if path **u** is α-optimal any other path **v** must incur a cost at least as high as **u** for every region between two recombination breakpoints x and y of path **u**. Otherwise it pays off to recombine from the red path to include the blue path between x and y.

Proof:

First, assume that $\varphi_\alpha(\mathbf{u},x+1,y)\leq\varphi_\alpha(\mathbf{v},x+1,y)$ for all $x,y\in\Delta(\mathbf{u})\cup\{0,N\}$ with $x<y$ and all paths **v**. It is easy to show that path **u** is α-optimal: with $x=0$ and $y=N$ it follows that $\varphi_\alpha(\mathbf{u},1,N)\leq\varphi_\alpha(\mathbf{v},1,N)$ for all paths **v**.

Now assume that **u** is α-optimal. Let $x,y\in\Delta(\mathbf{u})\cup\{0,N\}$ with $x<y$ and let **v** be an arbitrary path. We want to show that $\varphi_\alpha(\mathbf{u},x+1,y)\leq\varphi_\alpha(\mathbf{v},x+1,y)$ and make a case distinction depending on x and y.

a) $x=0$ and $y=N$: The desired result follows because **u** is α-optimal and therefore $\varphi_\alpha(\mathbf{u},1,N)\leq\varphi_\alpha(\mathbf{v},1,N)$.

b) $x=0$ and $y\neq N$: Define a path $\mathbf{k}=(v_1,\ldots,v_y,u_{y+1},\ldots,u_N)$ and decompose it into

$$\begin{aligned}\varphi_\alpha(\mathbf{k},1,N)&=\varphi_\alpha(\mathbf{v},1,y)+\varphi_\alpha(\mathbf{u},y+1,N)+(1-\alpha)r(y)\delta(v_y,u_{y+1})\\ \varphi_\alpha(\mathbf{u},1,N)&=\varphi_\alpha(\mathbf{u},1,y)+\varphi_\alpha(\mathbf{u},y+1,N)+(1-\alpha)r(y)\delta(u_y,u_{y+1})\end{aligned}.$$

As **u** is α-optimal we know that $0\leq\varphi_\alpha(\mathbf{k},1,N)-\varphi_\alpha(\mathbf{u},1,N)$ and it follows that

$$0\leq\varphi_\alpha(\mathbf{k},1,N)-\varphi_\alpha(\mathbf{u},1,N)=\varphi_\alpha(\mathbf{v},1,y)-\varphi_\alpha(\mathbf{u},1,y)+(1-\alpha)r(y)\Big(\delta(v_y,u_{y+1})-\delta(u_y,u_{y+1})\Big).$$

With $\alpha\in[0,1]$, $\delta(u_y,u_{y+1})=1$ and $\delta(i,j)\leq 1$ for all i we get

$$(1-\alpha)r(y)\Big(\delta(v_y,u_{y+1})-\delta(u_y,u_{y+1})\Big)\leq r(y)\Big(\delta(v_y,u_{y+1})-1\Big)\leq 0$$

leading to the desired result

$$\begin{aligned}0&\leq\varphi_\alpha(\mathbf{v},1,y)-\varphi_\alpha(\mathbf{u},1,y)+(1-\alpha)r(y)\Big(\delta(v_y,u_{y+1})-\delta(u_y,u_{y+1})\Big)\\ &\leq\varphi_\alpha(\mathbf{k},1,y)-\varphi_\alpha(\mathbf{u},1,y)\end{aligned}.$$

c) $x\neq 0$ and $y=N$: This case is analogous to **b)** using a path $\mathbf{k}=(u_1,\ldots,u_x,v_{x+1},\ldots,v_N)$.

d) $x\neq 0$ and $y\neq N$: The basic idea is the same as in case **b)**. Define a path $\mathbf{k}=(u_1,\ldots,u_x,v_{x+1},\ldots,v_y,u_{y+1},\ldots,u_N)$ and decompose the costs of paths **k** and **u** into

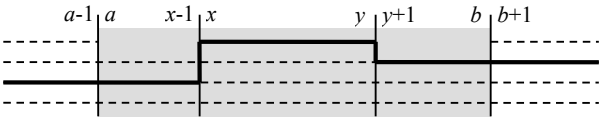


Figure 6.10: The topmost sequence contains a gap from position *a* to *b*. The gap region is also visualized with a grey background. The black path only covers the gapped sequence from position *x* to *y* and therefore recombines to include only the gap or part of it. Such a path does not make sense biologically as we only recombine to explain the nucleotides in the query sequence using indels. Lemma 4 provides a necessary condition to prevent this artifact.

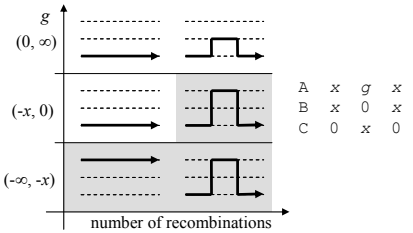


Figure 6.11: A grey background marks the α-optimal paths that we would like to avoid. The paths shown in the upper and in the middle row are both α-optimal for g=0 and setting g=0 corresponds to the case $G_\alpha(\mathbf{u},x,y)=B_\alpha(\mathbf{u},x,y)$. All paths in the right column are α-optimal for $x\geq 0$ and g=0 and one path also includes only the gap. Fortunately, both paths predict recombination breakpoints at the same position, such that this case is not as harmful. This observation actually holds for arbitrary alignments if we penalize gaps by $G_\alpha(\mathbf{u},x,y)=B_\alpha(\mathbf{u},x,y)$ but it is not proven here.

$$\varphi_\alpha(\mathbf{k},1,N)=\varphi_\alpha(\mathbf{u},1,x)+\varphi_\alpha(\mathbf{v},x+1,y)+\varphi_\alpha(\mathbf{u},y+1,N)+(1-\alpha)\left(r(x)\delta(u_x,v_{x+1})+r(y)\delta(v_y,u_{y+1})\right)$$
$$\varphi_\alpha(\mathbf{u},1,N)=\varphi_\alpha(\mathbf{u},1,x)+\varphi_\alpha(\mathbf{u},x+1,y)+\varphi_\alpha(\mathbf{u},y+1,N)+(1-\alpha)\left(r(x)\delta(u_x,u_{x+1})+r(y)\delta(u_y,u_{y+1})\right).$$

It follows that

$$0\leq\varphi_\alpha(\mathbf{k},1,N)-\varphi_\alpha(\mathbf{u},1,N)$$
$$=\varphi_\alpha(\mathbf{v},x+1,y)-\varphi_\alpha(\mathbf{u},x+1,y)+(1-\alpha)\left[r(x)\left(\delta(u_x,v_{x+1})-\delta(u_x,u_{x+1})\right)+r(y)\left(\delta(v_y,u_{y+1})-\delta(u_y,u_{y+1})\right)\right]$$

and we can perform the same approximation using $\alpha\in[0,1]$, $\delta(u_x,u_{x+1})=\delta(u_y,u_{y+1})=1$ and $\delta(i,j)\leq 1$ for all i and j:

$$(1-\alpha)\left[r(x)\left(\delta(u_x,v_{x+1})-\delta(u_x,u_{x+1})\right)+r(y)\left(\delta(v_y,u_{y+1})-\delta(u_y,u_{y+1})\right)\right]$$
$$\leq r(x)\left(\delta(u_x,v_{x+1})-1\right)+r(y)\left(\delta(v_y,u_{y+1})-1\right)\leq 0$$

and obtain the desired result:

$$0\leq\varphi_\alpha(\mathbf{k},1,N)-\varphi_\alpha(\mathbf{u},1,N)$$
$$\leq\varphi_\alpha(\mathbf{v},x+1,y)-\varphi_\alpha(\mathbf{u},x+1,y).$$

□

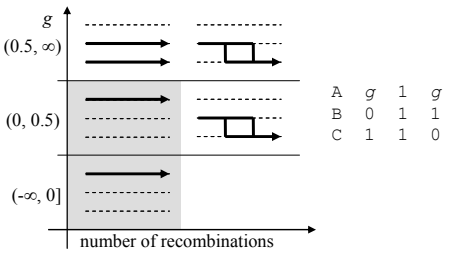


Figure 6.12: Many separate gaps in a sequence can cause gap costs that are systematically too low if the gap costs are derived from $B_\alpha(\text{-},x,y)$ or $B_\alpha(\mathbf{i},x,y)$. Sequence A has a mutation in the middle of the alignment and fits as well as sequences B and C outside the gap region. Consequently, an α-optimal path should not cover any part of sequence A, as A is not advantageous compared to the other sequences and as the α-optimal path should not include a gap only. But the lower bounds for the gaps are $B_\alpha(\text{-},x,y)=B_\alpha(\mathbf{i},x,y)=0$ and it is sufficient to set $g<0.5$ to obtain an α-optimal path that only covers sequence A. This artifact does not occur if we score gaps equal to or above $B_\alpha^{all}(-,x,y)$. If α is small, the path $\mathbf{u}^*$ is one of the paths in the right column and both gaps incur a cost of $g>0$. Thus, sequence A incurs a higher cost than any path in the right column and is not α-optimal. If α is large, the path $\mathbf{u}^*$ is one of the paths in the upper left of the plot: one gap then incurs a cost of $g>1$ and the other gap uses $g>0$. Again, sequence A is not part of any α-optimal path.

Lemma 4: Suppose there is a gap from position a to b in sequence i. Assume there exists x and y with $a\le x\le y\le b$ and a path $\mathbf{u}$ that covers sequence i only from position x to y (see Figure 6.10). In other words, $\mathbf{u}$ satisfies $u_p=i$ for all positions $p\in[x,y]$ and $x\text{-}1,y\in\Delta(\mathbf{u})\cup\{0,N\}$ such that x-1 and y are either recombination breakpoints or the boundary of the alignment. A sufficient condition and for some alignments also necessary condition that path $\mathbf{u}$ is not α-optimal is $G_\alpha(\mathbf{u},x,y)>B_\alpha(\mathbf{u},x,y)$.

Proof:

It is important to realize that the recombination breakpoint of path $\mathbf{u}$ is x-1 and not x. By construction of path $\mathbf{u}$ we know that $\varphi_\alpha(\mathbf{u},x,y)=G_\alpha(\mathbf{u},x,y)$ as there is no recombination event between x and y in path $\mathbf{u}$. Lemma 3 is applicable because $x\text{-}1\in\Delta(\mathbf{u})\cup\{0\}$, $y\in\Delta(\mathbf{u})\cup\{N\}$ and $x\text{-}1<y$. From Lemma 3 it then follows that path $\mathbf{u}$ is α-optimal only if $\varphi_\alpha(\mathbf{u},x,y)=G_\alpha(\mathbf{u},x,y)\le\varphi_\alpha(\mathbf{v},x,y)$ for any path $\mathbf{v}$. This condition must also be fulfilled for all paths $\mathbf{v}\in P(\mathbf{u},x,y)$ and therefore $G_\alpha(\mathbf{u},x,y)\le B_\alpha(\mathbf{u},x,y)$. In other words, a necessary condition for path $\mathbf{u}$ to be α-optimal is $G_\alpha(\mathbf{u},x,y)\le B_\alpha(\mathbf{u},x,y)$ and therefore $G_\alpha(\mathbf{u},x,y)>B_\alpha(\mathbf{u},x,y)$ is a sufficient condition to prevent such a path $\mathbf{u}$. Finally, the example in Figure 6.11 illu-

strates that the condition $G_\alpha(\mathbf{u},x,y) \leq B_\alpha(\mathbf{u},x,y)$ is sufficient to allow for a path like **u** at least for some alignments and some α.

□

6.3.4 Towards a Reasonable Gap Cost Function

If we combine the bounds of Lemma 2 and Lemma 4, we obtain

$$B_\alpha(\mathbf{i},x,y) < G_\alpha(\mathbf{i},x,y) \leq B_\alpha(\mathbf{i},x,y) + (1-\alpha)\big(r(x-1)+r(y)\big)\,. \tag{6.11}$$

These bounds already represent a big step towards treating gaps appropriately in Recco. The lower bound is also very intuitive: a gap should never incur a cost less than the least-cost alternative that avoids the gap. It is important to understand that this property depends critically on bounds for gap costs that change with α. Gap costs equal to the lower bound are probably also acceptable and are more practical to implement. The only restriction is that α-optimal paths should never predict a recombination breakpoint between two indels, i.e. inside a gap region. We can enforce this restriction for example by altering the set of permitted transitions in the dynamic program for each site. Furthermore, the bounds suggest a simple strategy for increasing the penalty of a gap in analogy to the strategy proposed in section 6.2.2.4. We could first use a gap cost at or close to the lower bound and then slowly increase the cost of the gap towards the upper bound while observing changes in the α-optimal path.

But there are still many open questions: what is the impact on the analysis if there are several gaps in the alignment, for example if there are two gaps at the same position or if there are two gaps in the same sequence in succession? Is there a simple gap cost function that satisfies these bounds? It is often difficult to provide a definite answer to these questions. The following exposition is therefore not exact in mathematical terms and is rather a guide for further investigations.

a) What happens if there are several indels or gaps at a position? In essence, the analysis is still valid although it does not treat this case explicitly. The main problem is that the lower bound for a gap could refer to another gap and introduce a cyclic reference. To be more specific, suppose that there is a gap in sequences *i* and *j* covering positions *x* to *y*. In this case, we could set $B_\alpha(\mathbf{i},x,y)=B_\alpha(\mathbf{j},x,y)$ if we allow scoring gaps at the lower bound. As a result, the cost of the gaps would be arbitrary. There is a simple strategy to avoid this problem. We can use the lowest cost of a path avoiding all indels, $B_\alpha(\text{-},x,y)$, instead of $B_\alpha(\mathbf{i},x,y)$.

b) What is the impact of a sequence that contains several gaps in succession? In this case, gap costs close to $B_\alpha(\text{-},x,y)$ or $B_\alpha(\mathbf{i},x,y)$ can actually result in gap costs that are systematically too low. If there are many separate gaps in succession, we choose the best alternative sequence or path as a basis for the cost of each gap but do not penalize if we choose a different alternative for each gap. As a result, the α-optimal path may erroneously favor the gapped sequence (see Figure 6.12). We can penalize selecting a different al-

ternative for each gap with a simple strategy: determine the minimum cost path **u*** that does not cover any indels and impute the cost of the gaps from **u*** (see section 6.3.1 for the exact definition of **u***). In other words, we use the cost $B_\alpha^{all}(-,x,y)$ as the lower bound for gaps. This strategy prevents the artifact shown in Figure 6.12. However, it would be desirable to show that the strategy also prevents this artifact for arbitrary alignments.

c) Does the column permutation test still work correctly? We have at least two options for assigning gap costs. We can assign gap costs for the original alignment once and then use the assigned gap costs for the column permutations. Alternatively, we can determine the gap costs for the original alignment and for each column permutation separately. It seems that the first option is preferable, as we would assign gap costs only once and would not score gaps differently for each column permutation.

d) Is there a gap cost function that satisfies the bounds? Obviously, we also have to show that there is a function satisfying the bounds. We would ideally want to score gaps as close to $B_\alpha^{all}(-,x,y)$ as possible and then slowly increase the gap costs observing changes in the α-optimal path. Figure 6.13 illustrates how gap costs close to the lower bound could be implemented. The basic idea is to assign the mutation cost of **u*** at position p to an indel at position p. As recombination cost occurs between two positions, we have to distribute the cost for a recombination event at position p in **u*** to the neighboring positions p and p+1.

e) Can we justify that a gap is incorporated into the α-optimal path if gaps are scored at or above the bound $B_\alpha^{all}(-,x,y)$? Lemma 5 below formalizes an interesting observation for these gap costs (also see Figure 6.14). Suppose that the α-optimal path **v** covers a gap. Any path that extends path **u*** around the gap incurs a higher cost than path **v** outside the gap region. In other words, the gap is incorporated into the α-optimal path because it is cheaper outside the gap region and not because the gap costs are excessively low. Unfortunately, this observation is only true for paths that extend path **u***. There can be other paths that incur a lower cost than **v** outside the gap region and a higher cost inside the gap region such that the argument is invalidated partly.

f) Can we still compute the Savings value? The analysis of Recco depends critically on the Savings value. Computing the Savings value requires that we can compute the derivative of φ_α with respect to α and therefore also the derivative of G_α. The implementation of gap costs proposed in Figure 6.13 clearly allows computing the derivate of G_α with respect to α. However, the path **u*** may change with α such that the derivative of G_α also changes with α. As a result, the parametric analysis described in section 5.4.2 does not apply anymore and needs to be modified. Even worse, it may be difficult to interpret the Savings value in the presence of gaps. A recombination event can avoid gap costs and not only mutation cost. But gaps also involve a cost term similar to recombination cost. The question is therefore: does such a recombination event avoid recombination cost? This complicates the analysis considerably.

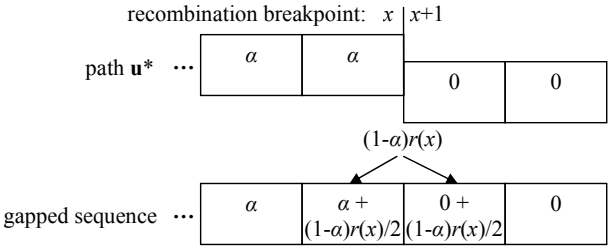


Figure 6.13: Each nucleotide of path **u*** is represented by a rectangle and each rectangle shows the mutation cost with respect to the query sequence. Path **u*** also has a recombination breakpoint at position *x*. The gapped sequence has indels at all shown positions. We can infer appropriate gap costs for the gapped sequence based on path **u***: assign the mutation cost of **u*** to the same position of the gapped sequence and distribute the cost for recombination in **u*** to the neighboring positions of the gapped sequence.

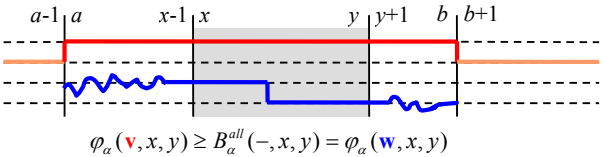


Figure 6.14: The plot illustrates the argument of Lemma 5. The topmost sequence contains a gap from position *x* to *y*. The blue path **w** represents an arbitrary path that is equal to **u*** in the gap region and for one nucleotide surrounding the gap. The red path **v** is *α*-optimal and covers the gap. Lemma 5 shows that the costs of path **v** outside the gap region are smaller than the costs of path **w**. The argument is simple. Path **v** is *α*-optimal and incurs less cost than any other path between *a* and *b*. But path **w** costs less or equal than **v** inside the gap region therefore incurs more cost than **v** outside the gap region.

Lemma 5: Suppose there is a gap in sequence *i* from position *x* to *y* and an *α*-optimal path **v** that covers sequence *i* from *a* to *b* with $a \leq x < y \leq b$ (see Figure 6.14). Assume that we score gaps at or above the lower bound such that $B_\alpha^{all}(-,x,y) \leq G_\alpha(\mathbf{i},x,y)$. It follows that any path **w** with $w_p = u^*_p$ for all $p \in [x\text{-}1, y\text{+}1]$ (i.e. there is no recombination event at positions *x*-1 and *y*) satisfies:

$$\varphi_\alpha(\mathbf{v},x,y) \geq \varphi_\alpha(\mathbf{w},x,y)$$

and

$$\varphi_\alpha(\mathbf{w},a,x-1) + \varphi_\alpha(\mathbf{w},y+1,b) \geq \varphi_\alpha(\mathbf{v},a,x-1) + \varphi_\alpha(\mathbf{v},y+1,b)$$

In other words, path **v** incurs a lower cost outside the gap region than any path **w** that covers path **u*** between positions *x* and *y* if **w** also has no recombination breakpoints at positions *x*-1 and *y*.

Proof:

The first inequality follows as path **v** covers a gap from positions *x* to *y* and as **w** is equal to **u*** between *x* and *y*:

$$\varphi_\alpha(\mathbf{v},x,y) \geq B_\alpha^{all}(-,x,y) = \varphi_\alpha(\mathbf{w},x,y)\,.$$

The second inequality requires that we decompose paths **v** and **w**:

$$\begin{aligned}\varphi_\alpha(\mathbf{v},a,b) &= \varphi_\alpha(\mathbf{v},a,x-1) + \varphi_\alpha(\mathbf{v},x,y) + \varphi_\alpha(\mathbf{v},y+1,b)\\ \varphi_\alpha(\mathbf{w},a,b) &= \varphi_\alpha(\mathbf{w},a,x-1) + B_\alpha^{all}(-,x,y) + \varphi_\alpha(\mathbf{w},y+1,b)\end{aligned}.$$

As path **v** is α-optimal and because of Lemma 3 it follows that

$$0 \leq \varphi_\alpha(\mathbf{w},a,b) - \varphi_\alpha(\mathbf{v},a,b)$$

and therefore

$$\begin{aligned}0 &\leq \varphi_\alpha(\mathbf{w},a,b) - \varphi_\alpha(\mathbf{v},a,b)\\ &= \varphi_\alpha(\mathbf{w},a,x-1) - \varphi_\alpha(\mathbf{v},a,x-1) + B_\alpha^{all}(-,x,y) - \varphi_\alpha(\mathbf{v},x,y) + \varphi_\alpha(\mathbf{w},y+1,b) - \varphi_\alpha(\mathbf{v},y+1,b)\\ &\leq \varphi_\alpha(\mathbf{w},a,x-1) - \varphi_\alpha(\mathbf{v},a,x-1) + \varphi_\alpha(\mathbf{w},y+1,b) - \varphi_\alpha(\mathbf{v},y+1,b)\end{aligned}$$

and the desired result follows by rearranging the terms:

$$\varphi_\alpha(\mathbf{v},a,x-1) + \varphi_\alpha(\mathbf{v},y+1,b) \leq \varphi_\alpha(\mathbf{w},a,x-1) + \varphi_\alpha(\mathbf{w},y+1,b)\,.$$

□

6.4 Outlook

We have analyzed common strategies for treating gaps in phylogenetics and in recombination analysis and provided an extensive discussion of their implications. Treating gaps as missing data is difficult for Recco as well as for several other methods for recombination detection. Simple strategies often discard a lot of information or even result in falsely detected recombination events.

We have proposed a simple and practical strategy that treats an indel as a fifth nucleotide and varies the cost of an indel. The strategy processes the output of Recco semi-automatically and detects artifacts in the output. Uncertain recombination events can be consolidated and analyzed manually.

We have also analyzed possible artifacts in the output of Recco and motivated several properties of reasonable gap costs for Recco. This analysis is an important step towards treating gaps appropriately in Recco and already restricts the space of potential gap cost functions considerably. Several properties of these gap costs are very attractive, for example that we never incorporate a gap without any surrounding nucleotides into the α-optimal path and that we can motivate if gaps are part of the α-optimal path. However, there are many aspects that are not studied in detail yet. It is particularly disturbing that it is much more difficult to interpret the Savings value for the proposed gap costs. Eventually, this analysis may lead to a strategy that treats gaps as missing data in Recco.

7 Validation

Validation is difficult for recombination detection programs and even harder for programs finding the recombinant sequences. One reason is that real alignments either show a recombination signal that is easy to detect, or that there is still uncertainty about whether recombination actually took place (Posada 2002). In comparison, simulated alignments have many advantages, such as an unlimited amount of alignments and total control over the simulated recombination scenario. Section 7.1 validates Recco on two types of simulated alignments: alignments corresponding to a fixed ARG and alignments corresponding to random ARGs drawn from the coalescent. The first approach allows for studying the power of identifying recombinant sequences or recombination breakpoints, as both are known for a fixed ARG. This approach is similar to the analysis of the example alignment in chapter 5. But we also study the effect of noise introduced by the mutation process and examine one thousand alignments for each ARG. The second approach works well for comparing different methods regarding their power of detecting recombination. Finally, section 7.2 validates Recco on real alignments from HERVs and HIV-1. The recombination signal is also discussed in more detail.

7.1 Validation on Simulated Alignments

Validating a method on many simulated alignments is indispensable, as it ensures that the method works well for arbitrary scenarios and as it allows for a quantitative comparison with other methods. We can only know the ancestry of a set of sequences and the corresponding mutation and recombination events with certainty for simulated alignments. But we can also test the method for many different simulation parameters and possibly uncover distinct strengths or weaknesses of the method. The only risk is that the simulation model is unrealistic and that the results do not transfer to real alignments. Fortunately, the methods for simulating sequence alignments are already quite mature and produce alignments close to the ones seen in practice.

7.1.1 Parameter Settings for Recco and Other Methods

The same parameter settings for Recco are used throughout the validation process. In particular, Recco uses Hamming distances for mutation and recombination cost, and constructs the full recombination event list for each alignment. *P*-values are estimated from 100 column permutations of the alignment and *p*-values below 0.05 are regarded as significant. Depending on the objective, the *p*-value for recombination in the alignment, for recombination in a sequence or for a breakpoint in the alignment is used. *P*-values are not corrected for multiple testing.

The methods used for comparison are Geneconv, PHI, NSS, MC^2 and RecPars (for detailed discussions refer to chapter 4). These methods dominate any other method regarding the power of detecting recombination in the power study (Posada and Crandall 2001),

unless the scaled mutation rate θ is lower or equal to ten. In other words, there is no other method with a better recombination detection performance for any setting of ρ and θ, if θ>10. A scaled mutation rate θ smaller or equal to 10 corresponds to an alignment where the two most diverged sequences only differ by about 20 mutation events. Obviously, it is not only difficult but sometimes also futile to detect recombination events in these scenarios as the number of mutations often does not allow for robustly inferring recombination. The parameter settings for the methods were usually chosen as in (Posada and Crandall 2001). The optimal parameters for Geneconv depend on the validation scenario. Only the results for the best parameters of Geneconv are reported in the following. For the fixed recombination scenarios, Geneconv performs better if it penalizes mismatches by x=-1. For random ARGs, Geneconv achieves a higher power if it disallows mismatches (x=-∞) and only searches for regions of maximum length where two sequences are equal. Even though the power of this setting is higher than for $x = -1$, it often finds significant regions that do not correspond to the regions exchanged during recombination. Recombination in an alignment was detected if Geneconv returned at least one fragment with a significant p-value. The implementation of PHI also includes the methods NSS and MC^2. The implementation only returns a p-value for recombination in the alignment and does not provide details on the recombination signal. PHI uses the default window size of 100 nucleotides and was set to perform 1,000 column permutations for computing p-values. Finally, RecPars is available as a command-line program (http://www.daimi.au.dk/~compbio/recpars/recpars.html) and outputs the predicted trees for different regions in the alignment and the total mutation cost required. Recombination cost was set to two times the cost of a mutation. Instead of relying on the absolute output of RecPars as in (Posada and Crandall 2001), we perform 100 permutations of the original alignment and detect recombination if the total mutation cost is significantly lower than the total mutation cost of column permutations. As RecPars only outputs a set of trees, it is difficult to determine the recombinant sequence in an automated fashion, and RecPars is only used for detecting recombination in the alignment in the following.

There are several new methods besides PHI that were not compared in (Posada and Crandall 2001) and could outperform the methods used for comparison. However, these methods require much more computation time, are usually not available as a command line program and are therefore unsuitable for a power study. For example, methods that optimize the recombination breakpoints (see section 4.3.5) usually require tens of minutes or even hours for a single alignment. The validation in this section is based on more than 50,000 alignments, and the computing time would be infeasible even with a compute cluster.

7.1.2 Validation on Fixed Recombination Scenarios

A total of four recombination scenarios are analyzed in the following (see Figure 7.1). Scenarios (S1) and (S2) are generated using the example ARG introduced in chapter 5,

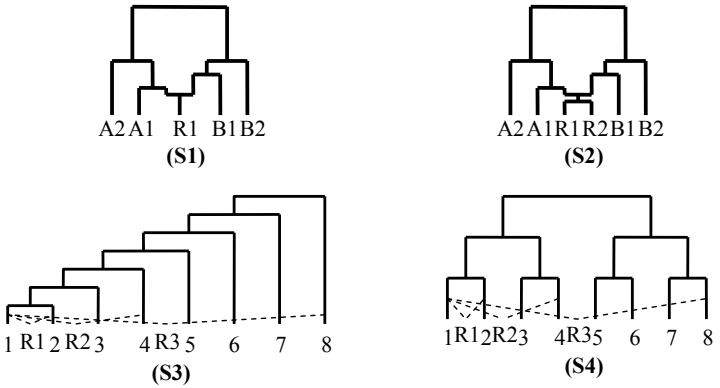


Figure 7.1: The genealogies used to simulate alignments for scenarios (S1), (S2), (S3) and (S4). Recombinations are marked with dotted lines for scenarios (S3) and (S4), as the ARG representation is not easy to draw for these ancestries. Recombination breakpoints always occur in the middle of the sequences. More details on the parameters can be found in Table 7.1.

where scenario (S1) does not include sequence R2. Scenarios (S3) and (S4) were originally introduced in (Holland, Huber et al. 2002). As all scenarios only contain recent recombination events, the recombination signal has a very simple structure and should be fairly easy to detect. (S1) and (S2) use the Jukes-Cantor model (JC) for the mutation process and (S3) and (S4) use the Kimura-two-parameter model (K2P) with a transition/transversion ratio of 2 (see section 3.2 for mutation models). 1,000 alignments were simulated using Seq-Gen (Rambaut and Grassly 1997) for each ARG and each setting.

7.1.2.1 Detecting Recombination in the Alignment

The power of detecting recombination is given in Table 7.1 along with the parameters of the simulation. For scenarios (S1) and (S2), alignments with two different sequence lengths were simulated to show the effect of recurrent mutations. The total number of expected mutations is the same and only the number of recurrent mutations differs between alignments of different length. Processing time for one alignment was less than one minute for any method and any scenario. Geneconv performs better if mismatches are allowed and penalized by -1 than if mismatches are disallowed. Only the power corresponding to this setting is shown in the table. If mismatches are not allowed, the power of Geneconv is much lower and the column would read 0.33, 0.42, 0.48, 0.58, 1.00, 0.98.

It is surprising that all five methods used for comparison are strongly affected by recurrent mutations. The power of detecting recombination increases from about 50% to 70–80% for Geneconv and MC^2 if we compare the different sequence lengths of scenario (S1) and (S2). Recurrent mutations affect the compatibility methods PHI and NSS to an

Scenario	*N*= (nt)	Tree height	Mut. model	ts/tv ratio	Geneconv	MC[2]	PHI	NSS	RecPars	Savings	OptSavings	FirstAngle	MaxCost
(S1)	100	0.25	JC	-	0.49	0.52	0.49	0.48	0.24	0.86	0.89	0.03	0.54
(S2)	100	0.25	JC	-	0.51	0.60	0.50	0.44	0.28	0.42	0.92	0.01	0.08
(S1)	1000	0.025	JC	-	0.73	0.79	0.90	0.88	0.61	0.97	0.98	0.05	0.64
(S2)	1000	0.025	JC	-	0.76	0.84	0.93	0.81	0.61	0.64	0.99	0.01	0.10
(S3)	1000	0.15	K2P	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.30	0.91
(S4)	1000	0.15	K2P	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.90	0.99

Table 7.1: The power of detecting recombination is shown for different methods and scenarios. For each scenario, 1,000 alignments were generated and then processed by different methods. Tree height is given as mutations per site from the root to a tip of the tree. ts/tv ratio refers to the transition to transversion ratio. The best value in each row is marked in red – OptSavings clearly outperforms all other methods.

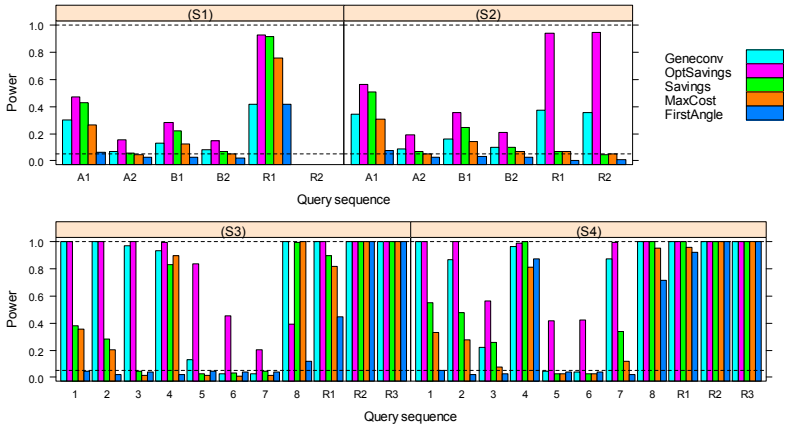


Figure 7.2: The probability of detecting a sequence as recombinant is shown for different methods and different scenarios. The significance level of 0.05 is marked by the dotted line at the bottom of the plots. **Top:** The results for scenarios (S1) and (S2) are only shown for a sequence length of 100 nucleotides. **Bottom:** The results for scenarios (S3) and (S4). See text for a detailed discussion.

even greater extent, as recurrent mutations reduce the compatibility between sites and compete with the signal for recombination. The power of detecting recombination increases from 50% to 90% in scenario (S1) and (S2) for PHI and NSS if the sequence length is increased. Scenario (S3) and (S4) are apparently too simple and do not pose a challenge for any recombination detection method.

The *p*-value for recombination based on the Savings value (shown in the Savings column) performs much better for (S1) than the methods used for comparison, but is worse for (S2). (S2) includes sequence R2 in the alignment, and the evidence for recombination is actually only derived from sequence A1 and not from the recombinant sequence R1 (see

Figure 7.2 for more details). The OptSavings value completely eliminates this deficit of the Savings value; the *p*-values corresponding to OptSavings show an excellent recombination detection performance for any scenario. Both Savings and OptSavings are also less affected by the amount of recurrent mutations than the other methods. The features FirstAngle and MaxCost perform worse than Savings or OptSavings. Only MaxCost can compete with Geneconv in scenarios (S1) and (S4).

7.1.2.2 Detecting Recombination in a Sequence

The same alignments as before were used to study the power of detecting recombination in a sequence. For scenarios (S1) and (S2) only the sequence length N=100 is analyzed. The same settings as before were used for Geneconv. Geneconv only detects recombination in sequence pairs, such that both sequences in the pair are considered as a recombinant.

Finding the recombinant sequence is more difficult (see Figure 7.2), particularly because it is very hard to discriminate between the true recombinant and sequences involved in the recombination. This is intrinsic to the problem itself and affects every method. In other words, many methods have a high probability of finding recombination in a parental sequence or a sequence closely related to the recombinant. The absolute feature value or the *p*-value may still suggest that the true recombinant shows a stronger recombination signal, though, but the strength of this effect cannot be determined from Figure 7.2.

Geneconv detects sequence A1 in scenario (S2) as a recombinant with almost the same probability as R1 and R2, probably because the pair of sequences often involves A1 as well. OptSavings has an even stronger tendency of detecting recombination in other sequences in (S3) and (S4): sequences 3, 5, 6 and 7 are not involved in a recombination but are still detected as a recombinant. The reason is that OptSavings removes parental sequences until other related sequences also benefit from recombination. It is possible that the absolute value of OptSavings for different sequences can discriminate between recombinant and non-recombinant sequences, but this is not visualized in the plots. In contrast, Savings only detects recombination in recombinants and their parental sequences for (S3). The true recombinant sequences show the highest probability, though. Savings detects some non-recombinants as recombinant for (S4), but the overall pattern is similar to (S3). MaxCost performs worse than Savings for detecting the true recombinant sequences and shows the same pattern detecting related sequences as a recombinant. Only FirstAngle shows a different pattern and just detects the true recombinants for (S1) and (S3). For (S4), only sequence 4 and 8 are classified as recombinant, too. However, it is unclear if this effect is mainly caused by the lower overall recombination detection performance of FirstAngle.

The results for scenarios (S1) and (S2) are as expected: OptSavings and Savings are very similar if there are no closely related sequences to the recombinant in the alignment. But for (S2) Savings does not detect recombination in the recombinants R1 and R2, and ex-

tracts the evidence for recombination solely based on sequences A1 and B1. In other words, Savings detects recombination in an alignment of (S1) only based on the recombination signal in A1 and B1. MaxCost and FirstAngle perform worse than Savings, but show the same trend as Savings.

In summary, OptSavings performs extremely well for recombination detection, but its interpretation for detecting recombinant sequences is unclear. Savings works better than OptSavings and Geneconv for detecting recombinant sequences, but does not find the recombinant if it is concealed by another closely related sequence. FirstAngle only has a poor overall detection performance, but appears to extract a different aspect of the recombination signal.

7.1.2.3 Detecting the Recombination Breakpoint

There are several methods for computing *p*-values for a recombination breakpoint in Recco. In the following, we only study the *p*-values for a breakpoint in the alignment, as the *p*-values for a breakpoint in the query sequence are very similar if the query is a recombinant. "Savings BP-pval" denotes the results based on $\Pr(X_{\cdot p} \geq S_{\cdot p})$ and compares the Savings value to the expected distribution of Savings at the same position. "Savings" corresponds to $\Pr(X \geq S_{\cdot p})$, or the alignment *p*-value of $S_{\cdot p}$, and compares the Savings value to the expected distribution of Savings in the alignment. The latter is more conservative and represents the output of the recombination event list L if only recombination events with a significant alignment *p*-value are retained in the list. Finally, we can also use OptSavings as the input for both types of *p*-values. The probability of detecting a recombination breakpoint for each position is shown for scenario (S2) in Figure 7.3.

Figure 7.4 focuses on the false detection rate for detecting the recombination breakpoint in scenario (S3). Other scenarios show a similar picture. Savings BP-pval and OptSavings BP-pval detect a recombination breakpoint at positions 1–400 in about 5% of the cases and control the false detection rate in principle. But this behavior is inacceptable in practice, as there are about 50 spurious recombination breakpoints detected for an alignment of length 1,000. Savings and OptSavings perform much better regarding the false detection rate. Consequently, the method of choice for detecting breakpoints is to construct the recombination event list and to discard all recombination events with a non-significant *alignment p*-value.

The implementation of Geneconv is the only method for comparison that conveniently outputs predicted recombination breakpoints and the associated *p*-values. A breakpoint for Geneconv was defined as the start or end position of a gene conversion fragment with a significant *p*-value. Consequently, the alignment boundaries are also detected as recombination breakpoints for Geneconv, but it would be easy to filter this effect. The result for detecting recombination breakpoints in scenarios (S1) and (S2) is shown in Figure 7.5. Geneconv has a smaller false detection rate than the Savings value of Recco, but also has a much lower power. Savings outperforms Geneconv even for scenario (S2). However,

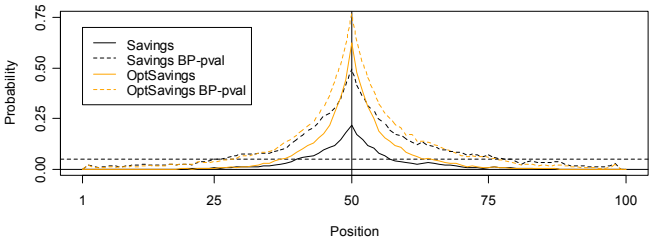


Figure 7.3: The probability of detecting a recombination breakpoint at a certain position for scenario (S2). The true breakpoint is marked by a black, vertical line. The legend is explained in the text. Apparently, Savings is more conservative than Savings BP-pval. It has a lower power of detecting the true recombination breakpoint, but controls the false detection rate much better. In other words, the peak that represents the true breakpoint is more localized for Savings and only positions 40 to 60 are detected as a breakpoint in more than 5% of the cases. As before, OptSavings avoids the limitation inherent to Savings and recovers the recombination signal for (S2). OptSavings and Savings are almost equal for scenario (S1), as sequence R2 is not included in the alignment (result not shown here).

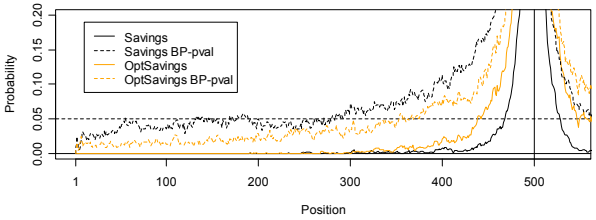


Figure 7.4: The probability of detecting a recombination breakpoint for scenario (S3). The notation is the same as for Figure 7.3. The complete plot is almost symmetric; positions 600-1000 are very similar to positions 1-400. Savings BP-pval and OptSavings BP-pval falsely detect a recombination breakpoint at each position in about 5% of the alignments. As a result, there are about 50 falsely detected breakpoints per alignment, which is clearly too high for practical settings. The result for scenario (S4) is very similar to (S3) and is not shown here.

Savings also shows a more dispersed profile overall and often detects recombination breakpoints around positions 40 and 60.

Figure 7.6 studies the cause for the dispersed profile of Savings in scenario (S3). As before, the result transfers to other scenarios as well. The method “Savings>50” only retains recombination events with a Savings value greater than 50. The 5% threshold intersects roughly at the same position as for Geneconv. However, the power of detecting the true recombination breakpoint for Savings>50 is much higher than for Geneconv. In other

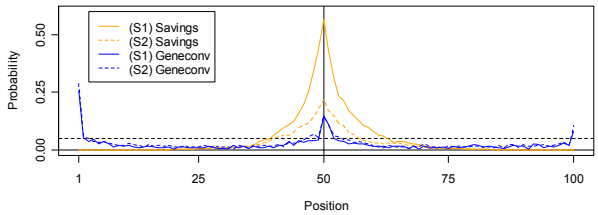


Figure 7.5: A comparison of Geneconv and Savings for scenarios (S1) and (S2). Geneconv does not suffer from the same limitation as Savings and the plots for (S1) and (S2) are almost identical. Nevertheless, Savings has a higher probability of detecting the true recombination breakpoint in both scenarios. On the other hand, Savings also shows a more dispersed profile and does not localize the breakpoint as well.

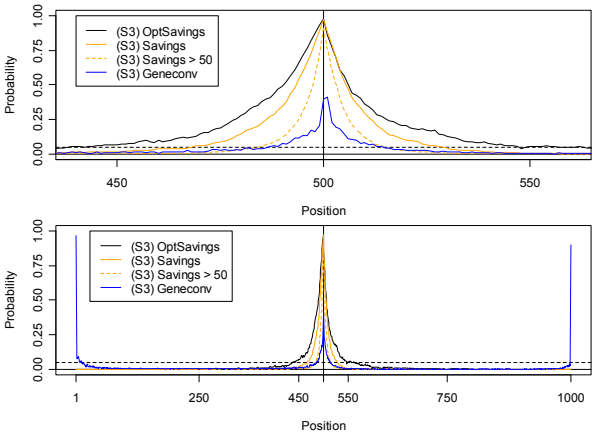


Figure 7.6: A comparison of different methods for scenario (S3). Savings>50 refers to recombination breakpoints that are significant and have a Savings value greater than 50. **(a)** Only the positions 450 to 550 are shown. Savings>50 and Geneconv show a similar dispersion for the breakpoint, but Savings has a much higher power. **(b)** The plot for the full alignment shows that all methods control the false detection rate nicely for positions further away from the true breakpoint.

words, ordering the recombination events by the Savings value helps to find more relevant accurate breakpoints first. It is also interesting that the probability of detecting the true recombination breakpoint is almost 100% for Savings and for Savings>50. Hence, the profile of Savings may be more dispersed because sequences other than the true recombinant show a recombination signal close to position 500 that is significant, but not as strong as the recombination signal from the true recombinant. The recombination break-

point for these other sequences may not be localized as well as for the true recombinant sequence and cause the observed dispersion.

7.1.3 Validation on Random ARGs

The validation on fixed recombination scenarios compares the different methods in detail and already provides a good idea on the performance that we expect for alignments of an unspecified scenario. However, we risk missing scenarios where Recco fails, as the validation does not cover a wide range of different ARGs. In particular, the recombination scenarios studied in the previous section only contain recent recombination events. As an alternative, we can draw random ARGs from the coalescent and simulate an alignment according to the ARG. The ARGs generated from the coalescent depend on several population genetic parameters (see section 3.3): the scaled mutation rate θ, the scaled recombination rate ρ and the number of sequences in the sample. Consequently, we can compute the power of detecting recombination in an alignment depending on θ and ρ. For a method the power of detecting recombination is the probability that the method returns a significant p-value for input alignments that contain recombination. Unfortunately, it is much more difficult to perform a detailed analysis in the case of random ARGs and extract the power of detecting the recombinant sequence or the recombination breakpoint. The problem is that the ARG may contain irrelevant recombination events (see section 3.3.3.4) or that it may be difficult to define the recombinant sequence accurately (see section 4.1).

7.1.3.1 Simulation Setup

The power study in the following uses similar parameters as the study by Posada and Crandall. All simulated alignments have a sequence length of N=1,000 nucleotides and a sample size of M=10 sequences. The program ms (Hudson 2002) was used to simulate 1,000 ARGs for each parameter combination with $\rho \in \{1, 4, 8, 16, 32\}$ and $\theta \in \{10, 20, 50, 100, 200\}$, resulting in a total of 25,000 ARGs. For each ARG, we have simulated one alignment with dawg (Cartwright 2005) using the Jukes-Cantor substitution model. As described in section 3.3.2, the parameter θ directly measures the expected number of mutations per genome from the root to a leaf of the simulated coalescent trees. The two most diverged sequences in an alignment therefore are separated on average by 20 to 400 mutations, depending on the value of θ.

The simulation procedure described above differs from the original publication of Recco (Maydt and Lengauer 2006), which relied on treevolve (Grassly, Harvey et al. 1999) for simulating alignments. There are several reasons why the transition to ms and dawg has been performed. First, ms and dawg can simulate alignments with a large amount of recombination and do not fail for ρ=32. Second, ms outputs the ARG and in principle allows for analyzing the types of the recombination events in more detail. Finally, dawg

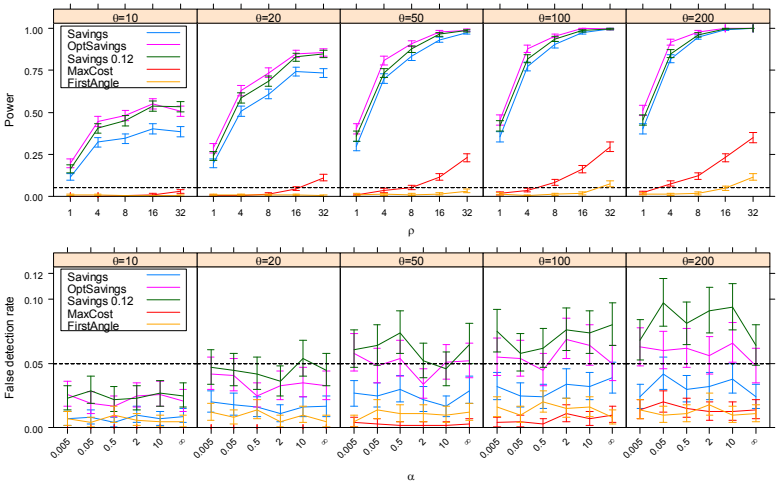


Figure 7.7: **Top:** The power of detecting recombination is plotted for Recco and different parameter combinations of θ and ρ. OptSavings dominates all other methods. **Bottom:** The false detection rate does not depend greatly on the level of rate variation α, but only increases with θ. The false detection rate of OptSavings is only slightly above 0.05. As expected, Savings 0.12 produces too many false positives.

can also simulate alignments with indels. The last two aspects are not relevant for the subsequent analysis and only allow for a more detailed analysis in the future.

The power of detecting recombination is only interesting if the recombination detection method also controls the false detection rate. A method should not erroneously detect recombination in more than 5% of the cases. The false detection rate is studied by simulating alignments with different mutation rates and rate variation among sites, but without recombination. The simulations were performed with the same approach as before. Rate variation was modeled using the gamma distribution with shape parameter α (Yang 1996). The shape parameter α is not related to the parameter α in the optimization problem – α only refers to the shape parameter in the following. A higher value of α represents a lower amount of rate variation and $\alpha=\infty$ denotes no rate variation at all. The parameters were varied such that $\rho=0$, $\alpha \in \{0.1, 0.5, 2, 20, \infty\}$ and, as before, $\theta \in \{10, 20, 50, 100, 200\}$. Even though this approach was used in (Posada and Crandall 2001) to determine the false detection rate of the methods, it is clear that the rate variation introduced in the alignments is quite artificial. Each site belongs to a certain rate category, but the rate of evolution is independent between sites. Rate variation therefore does not simulate mutation hotspots or regions that evolve faster than the rest of the gene. However, some methods such as GARD or PLATO detect recombination as a change in the local tree including branch lengths and consequently detect recombination at the boundaries of

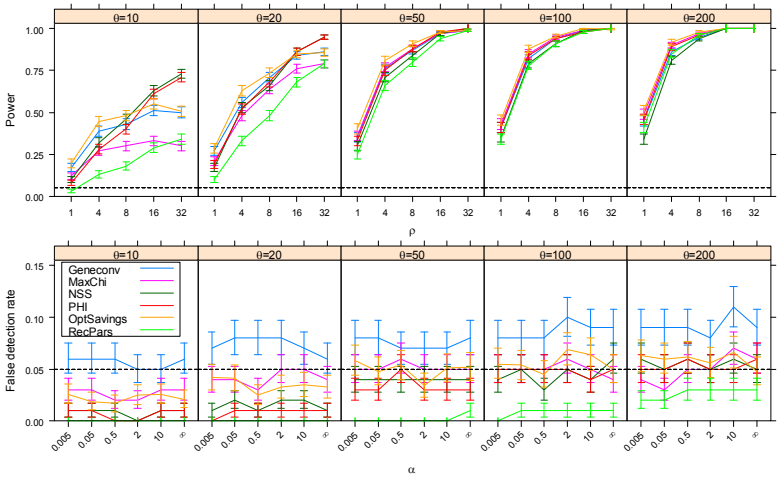


Figure 7.8: The same comparison as in Figure 7.7 is shown for a different selection of methods. The text provides a detailed discussion.

mutation hotspots. A more complete evaluation of the false detection rate should therefore also simulate mutation hotspots as in (Husmeier 2005; Husmeier, Dybowski et al. 2005). On these data, Recco is not adversely affected by mutation hotspots and does not generate an elevated false detection rate (not shown).

The recombination detection methods used for comparison were run with the same settings as described before (see section 7.1.1). For Recco, we can compute *p*-values for recombination in the alignment based on Savings, OptSavings, MaxCost and FirstAngle. The Recco publication (Maydt and Lengauer 2006) also defines a method called "Savings 0.12", which detects recombination if the *p*-value for recombination based on the Savings value is below 0.12. Hence, we expect to falsely detect recombination in about 12% of the cases for Savings 0.12. Savings 0.12 was designed to have a false detection rate comparable to Geneconv, such that the power of both methods can be compared more fairly. However, OptSavings performs better than both methods without changing the significance level and is preferable for a practical setting. Savings 0.12 is only presented to allow for a comparison with the original publication.

7.1.3.2 Results

Figure 7.7 compares the results for different methods in Recco. OptSavings performs better than Savings 0.12 and limits the false detection rate to about 5%. The MaxCost and FirstAngle features have a very low power of detecting recombination, but also show a lower false detection rate than other methods in Recco.

Figure 7.8 compares all other methods with OptSavings. It is interesting that the shape of the power curve also reflects the principle of the method. Recco, MC^2 and Geneconv analyze the distribution of mutations and can detect recombinations that do not change the unrooted topology of the local trees. On the other hand, they do not analyze local trees or infer more ancient evolutionary events. The other methods PHI, NSS and RecPars detect recombination by finding inconsistencies between local tree topologies, but are limited to detect recombinations that change the unrooted local tree topology. OptSavings, MC^2 and Geneconv work best for lower recombination rates such as $\rho \leq 8$. PHI and NSS outperform these methods if $\rho > 8$ and $\theta \leq 20$, but quickly loose their advantage as θ increases and the alignments contain more mutations. OptSavings consistently outperforms MC^2 and Geneconv for all parameter settings and is only inferior to PHI and NSS if $\rho > 8$ and $\theta \leq 20$. All methods but Geneconv limit the false detection rate to about 5%. Savings 0.12 is comparable to Geneconv regarding the power and the false detection rate (not shown here, see (Maydt and Lengauer 2006) for details). Savings performs worse than Geneconv, but controls the false detection rate to 5%. RecPars is an interesting case, as it has a much lower false detection rate than other methods. A careful selection of the parameters for RecPars may lead to a higher power for recombination detection at the cost of an increased false detection rate.

7.2 Validation on Real Alignments

Validating methods for recombination analysis on real alignments is difficult and error prone. There are two major deficits of validating methods on real alignments when compared to validation on fixed recombination scenarios. First, we only have a single alignment for each recombination scenario. Second, and much more critical, we never know the history of recombination and mutation events that generated the alignment. Some alignments and recombinant sequences have already been analyzed extensively and are fairly well understood, such as the dataset about HIV-1 subtyping in section 7.2.2. However, the analysis is often performed using popular methods, for example bootscanning (see section 4.3.4.1), and the results are manually curated afterwards. Consequently, the results are often biased by the method used for the initial analysis and cannot be regarded as a gold standard for recombination analysis.

Nevertheless, there are many good reasons to perform a validation on real alignments. Only real alignments contain all kinds of genomic variation and population genetic effects, for example gaps, repeats, migration, population bottlenecks, population growth or a limited amount of selection. Even very simple aspects of real alignments, such as gaps, have a significant impact on the analysis of recombination. Some methods for the analysis of recombination ignore gaps in alignments and do not treat them appropriately (see chapter 6.1). Finally, real alignments can also serve as instructive cases for improving a method and for determining the most relevant part of the output provided by the method.

The following section first presents an analysis of a new set of HERV sequences and then investigates a well-studied set of HIV-1 recombinant sequences. The task for both cases is to determine the composition of a potentially recombinant query sequence regarding a set of reference sequences. This is a common task in practice, as it is often assumed that a set of reference sequences is established in the population and new, potentially recombinant sequences are generated from these reference sequences. Even if intermediate sequences were generated, they would usually not harbor many mutations compared to the reference sequences and could be ignored in the analysis of the query sequence.

Consequently, we use Recco to estimate the composition of the query sequence and only use the recombination event list for the query sequence to scan for potential recombination events. To be more specific, we estimate the recombinant structure of the query sequence by using a sequence p-value cutoff of 0.05 in the recombination event list L_i (see section 5.6.2). In other words, we determine the α-value that corresponds to the last significant recombination event in L_i and then recover the α-optimal path for it. Scoring gaps appropriately is particularly challenging, and details are given in the respective analysis. In general, we check manually for artifacts caused by inappropriate gap scoring, for example if gaps attract the α-optimal path and should incur a higher cost.

We use the DNA mutation model of Recco to assign costs to mutation events. The DNA mutation model is a simple extension of Hamming distances allowing for ambiguous DNA nucleotide symbols. A character in the alignment is best represented as a set of nucleotides in this case and we can compute the cost of a mutation between two sets of nucleotides a and b as

$$m(a,b) = 1 - \frac{a \cap b}{a \cup b}. \tag{7.1}$$

Hence, the mutation cost is equal to hamming distances if there are no ambiguous nucleotide symbols in the alignment.

7.2.1 Analysis of Expressed HERV-K(HML-2) Sequences

As described in section 2.2, HERVs, or human endogenous retroviruses, amplified millions of years ago by reintegrating into the human genome over and over again and created several evolutionarily related HERV families. In other words, the HERV sequences encoded in the human genome of one individual are usually related by some sort of ancestry, for example a phylogenetic tree if recombination does not occur. As reintegration is a rare event today, most HERV loci are shared among the individuals in the human population. The genetic sequence of a HERV locus may differ slightly between individuals in the population, just as any part in the human genome can differ between individuals. The variation between different HERV loci is usually larger than the variation for the same HERV locus between different individuals. As a consequence, there are mainly two types of analyses: we can either study the long-term evolution of HERVs and HERV families or the short-term evolution of HERVs. The long-term evolution is already adequately covered by comparing different HERV loci within one individual. However, if

we are interested in the short-term evolution of HERVs, we have to sequence one HERV locus in several individuals or scan for new HERV loci in the genome that have a similar sequence. If the HERV locus evolved even in the recent past, we would expect to recover similar HERV sequences between individuals at different loci in the genome, or we may even find that the HERV sequence is missing altogether in some individuals.

The following section focuses on an exceptionally young HERV family, HERV-K(HML-2) or HML-2 in short and summarizes an analysis of HML-2 sequences using Recco. HML-2 is particularly interesting, as proviruses of the HML-2 family are still transcribed in several human tissues and encode many functional viral proteins, such as gag, pol and env. Based on a sequence analysis, Belshaw and coworkers (Belshaw, Pereira et al. 2004) found that the env gene is under purifying selection and proposed that several HML-2 proviruses could still be infective today. Even though some of these proviruses appear to be fully intact, an infectious variant has not been isolated so far. Detecting an infectious and actively replicating HML-2 variant would therefore be a significant finding.

7.2.1.1 Sequence Data

The input sequences for the analysis with Recco were chosen by a prefiltering step as described in the following. Flockerzi and coworkers (Flockerzi, Maydt et al. 2007; Flockerzi, Ruggieri et al. 2008) sequenced 642 HML-2 cDNA's with a length of about 800nt from the gag region during an HML-2 expression study. These sampled sequences were compared to 63 reference sequences, called references in the following. The references comprise the HML-2 proviruses published in the human genome reference sequence and a HML-2 provirus known to be missing therein, HERV-K113 (Turner, Barbulescu et al. 2001). An important aspect of the reference alignment is that some references corresponding to older HML-2 loci have a 96bp insertion in the gag gene.

The samples and the references were then combined in a multiple alignment and the closest reference for each sample was determined by Locus-Assigner. Locus-Assigner is a simple Python script that uses Hamming distance as a distance metric, treating gaps as a fifth character. Most samples could be assigned to one of the references with only few mutations. However, about 5% of the samples, called HERV-KX in the following, showed 13 or more nucleotide differences to the closest reference. Clearly, the number of nucleotide differences could be inflated, as gaps were treated as a fifth character and the presence or absence of a large 96bp insertion in a sampled sequence could already cause a high number of differences. The fundamental question remains the same, though: where do the HERV-KX sequences come from and how did they accumulate the mutations or indels? Are they indicative of an infectious and rapidly evolving HERV variant?

HERV-KX sequences represent HML-2 sequences that were not observed before. There are several possible explanations for the origin of these sequences. First, they could represent a set of yet undiscovered HML-2 variants that are fixed in the genome of a hu-

man subpopulation. However, it is very unlikely that so many HML-2 variants went unnoticed so far. Second, they could stem from an active HML-2 provirus that accumulates mutations *in vivo* or even produces recombinant sequences. In this case, we would expect that the HERV-KX sequences are isolated repeatedly from the same individual, in the same way as the HML-2 reference sequences were often detected in the sampled sequences. But the HERV-KX sequences were quite diverse and the same HERV-KX sequence was not identified twice. Finally, a third option is that the HERV-KX sequences are artifacts of the experimental procedure and were generated, for example, by *ex vivo* recombination. There are two well-known causes for *ex vivo* recombination during a sequencing procedure: during production of cDNA from the viral RNA by reverse transcriptase (RT) (Luo and Taylor 1990) and during amplification of cDNA by PCR (Meyerhans, Vartanian et al. 1990). Both processes are well known to cause recombination *ex vivo* and may complicate the analysis of sequences considerably. The hypothesis of *ex vivo* recombination is tested in the following with Recco by investigating the potentially recombinant origin of the HERV-KX sequences.

7.2.1.2 Methods – A Simple Strategy for Treating Gaps in Recco

The large gap region in the multiple alignment makes the analysis of HERV-KX sequences particularly challenging. As described in chapter 6, gaps can lead to several artifacts and require a careful analysis. The analysis of gaps in Recco was actually motivated by the present study of HERV-KX sequences. Hence, we could not rely on the analysis of chapter 6 back then and only had a limited number of options. We could either (i) discard sites that contain a gap character, (ii) treat each gap character as a fifth nucleotide state or (iii) treat each consecutive run of columns containing gaps as a large polymorphism, as in Geneconv. All these options have certain disadvantages and may cause severe artifacts in the output. The first option results in an unacceptable loss of information, as the presence of the 96bp insertion already represents important information. More notably, different references that harbor the 96bp insertion also show a different sequence for the insertion. The second option may lead to an artificially high similarity or dissimilarity between sequences in gap regions and can produce spurious recombination events. Finally, the third option prohibits recombinations in any run of columns containing a gap. A sequence with a long gap may therefore confound recombinations in other sequences. It is also difficult to choose an adequate scoring term for the large polymorphisms that are generated by this procedure. In conclusion, these approaches for treating gaps either discard a lot of information and thus miss recombination events or may infer spurious recombination events solely based on gap information.

We decided to develop and implement a heuristic approach that may discriminate between spurious recombination events inferred because of gaps and recombination events inferred solely based on nucleotide polymorphisms. The basic idea is to treat gaps as a fifth nucleotide state, but vary the penalty for incorporating gap characters into the

α-optimal path and determine the recombination events that are stable over a large range of gap penalties. As described in chapter 6, we never penalize pairing a gap in the query sequence with a nucleotide in the explanation. In other words, all columns with a gap in the query sequence are effectively removed from the alignment. However, we penalize pairing a nucleotide in the query sequence with a gap character in the explanation, as this situation corresponds to missing information.

To capture the effect of the gap penalty on inferred recombination events, we execute Recco for all gap penalties g with $g \in \{0.016, 0.02, 0.04, 0.07, 0.1, 0.2, 0.4, 0.6, 0.8\}$. As gaps are strongly preferred for gap penalties below 0.016 and result in spurious recombination events, we do not consider gap penalties smaller than 0.016. The results are then automatically processed as follows: recombination events with a p-value above 0.05 are discarded as not significant. If a recombination event with the same start and end position has a significant p-value for all gap penalties, it is accepted as correct. Other recombination events are manually examined and consolidated. In total, we confirmed 17 recombination events automatically and analyzed 25 recombination events manually, most of which were verified as true recombination events.

It is clear that the same analysis could be performed more accurately and efficiently if the parametric analysis was extended to cover both parameters, α and g. However, such a parametric analysis is not only numerically difficult, but also results in a two-dimensional parametric cost surface. This cost surface is not as easy to interpret as the one-dimensional cost curve. Nevertheless, such an extended parametric analysis may become interesting as future work, if we understand the effects of varying g in more detail.

Scoring gaps as a fifth nucleotide mainly causes two artifacts: the α-optimal path may introduce additional recombination events either to incorporate a gap for low gap penalties or to avoid the gap for high gap penalties. These two artifacts may have several complex effects on the inferred recombination events, as described below, and the p-values of the inferred recombination events can vary widely for different gap penalties. These recombination events are called RiQ ("recombination event in question") in the following and require a manual analysis. The output of Recco for different gap penalties was extremely helpful for identifying the RiQs. There are several effects that may generate an RiQ:

1. The RiQ is directly caused by the gap.

As a result, the RiQ is only inferred for low or high gap penalties. This effect is the most frequent for generating RiQ. For example, sequence B282 in Table 7.3 shows a RiQ in path b1) that is only inferred for gap penalties below 0.023. The gap region in path b1) covers 397 positions. The alternative path b2) is only inferred for larger gap penalties and only incurs 9 mutations more than path b1). Consequently, path b2) is preferred for gap penalties above 9/397=0.023.

Query	Breakpoint	Gap Penalty	Savings	*P*-value	Caused by
B282	583	0.016-0.02	~18	0.001	1.
	238+	0.016	5.35	0.061	
		0.02	6.94	0.022	
		0.04-0.8	16	0.001	
	583+	0.016	5.35	0.061	2.
		0.02	6.94	0.022	
		0.04-0.8	8	~0.01	
B100	510+	0.016-0.2	5	<0.005	3.
		0.4-0.8	5	>0.2	
	164+	0.016	7.5	0.001	
		0.02	8	0.001	
		…	…	0.001	
		>0.45	49	0.001	

Table 7.2: This table shows part of the recombination event list for different gap penalties and provides *p*-values for the recombination breakpoints listed in Table 7.3. Some breakpoints are only inferred or significant for some gap penalties, and the relevant information is marked in red.

Query		Path	Breakpoints	#Recs	#Muts	#Gaps	Savings	Optimal for g
B282	a)	7_450	–	0	25	0	–	all[#]
	b1)	8_121, 3_385	583	1	0	397	25-397g	g<0.023
	b2)	1_504, 3_354	238+	1	9	0	16	g>0.023
	c)	1_504, 7_450, 3_385	238+, 583+	2	1	0	b1) 24-397g b2) 8	all[#]
B100	a1)	7_450	–	0	0	54	–	g<0.45
	a2)	3_385	–	0	96	11	–	g>0.45
	b)	7_450, 3_385	164+	1	5	0	a1) 54g-5 a2) 91+11g	all[#]
	c)	7_450, 3_385, 7_345	164+, 510+	2	0	0	5	all[#]

[#] for all gap penalties studied in the analysis, i.e. $g \in [0.016,0.8]$

Table 7.3: The optimal paths are shown for different α and gap penalties g. The column Path describes the sequences involved in the recombination, Breakpoints the breakpoint location and #rec, #muts, #gaps column shows the number of recombinations, mutations and gaps in the path. For example, path b1) for the query B282 uses sequence 8_121 up to position 583 and then switches to sequence 3_385. Path b1) requires one recombination, no mutations and 397 gap characters and is only α-optimal for gap penalties below 0.023. The Savings value corresponding to path b1) is 25-397g and varies with the gap penalty g. The Savings value for path c) also depends on the gap penalty: path b1) is only used for comparison if g<0.023 and the Savings value is therefore (25-397g-1)/(2-1).

2. The RiQ is indirectly caused by a gap in the α-optimal path with less recombination.

The Savings value of a recombination event always compares the mutation and recombination cost of two paths: the α-optimal path including the recombination event and the alternative α-optimal path excluding the recombination event (for a different α). Hence, the Savings value and the *p*-value of the RiQ may change simply because the alternative path covers a gap. A simple example is path b) of sequence B100 (see Table 7.3). The path without recombination is a1) for gap penalties below 0.45. The mutation cost of the path increases with increasing gap penalty, because we treat gaps as a fifth nucleotide state. Consequently, the Savings value of the recombination event in path b) also varies for different gap penalties below 0.45, as shown in the column "Savings".

3. The RiQ is indirectly caused, as more complex effects affect the *p*-value of the RiQ.

A recombination event may also become insignificant for some gap penalties because the distribution of $X_{i\cdot}$ changes with the gap penalty. In other words, the gap penalty influences the distribution of Savings values in column permutations of the alignment. Consequently, a constant Savings value may correspond to different *p*-values for different gap penalties. An example is the recombination event at position 510 in path c) of sequence B100 (see Table 7.3). The recombination event always corresponds to a Savings value of 5, irrespective of the gap penalty. However, the distribution and the maximum of Savings values for column permutations changes for different gap penalties, just like the Savings value of path b) does. As a consequence, the Savings value of 5 is insignificant for gap penalties that are larger than ~0.3 (see Table 7.2).

7.2.1.3 Results and Discussion

Consolidating the results of the analysis involved a lot of manual work and was quite challenging, as we were unraveling the effects causing recombination events in question. Table 7.4 shows the results of the original publication (Flockerzi, Maydt et al. 2007). The heading 'Locus Assigner' lists the closest locus to the query sequence and the number of differences (mutations or gaps) to the closest locus. 'Recco Analysis' describes the α-optimal path for each query sequence deemed most likely by the manual analysis. For example, Recco predicts that sequence B260 is a recombinant of 7_450 and 5_544 with a breakpoint at position 164–181. The first part of B260 up to the breakpoint actually matches 7_450 perfectly and does not introduce any differences. The second part of B260 involves four differences to locus 5_544, such that there are only four differences in total. '/…' in the column Loci indicates that more than one locus fits equally well to the query. Finally, the list also provides the *p*-value of the path based on 1,000 column permutations using a gap penalty of 0.2. Some paths do not have a significant *p*-value but may still represent a good hypothesis, as Recco often predicts only few recombination events (see

Query Sequence	Locus Assigner		Recco Analysis				
	Closest Locus	#diff's	Loci	Breakpoints	#diff's to Locus	*P*-value	Total #diffs
B251	7_450	56	3_610/… 7_450 10_019 1_504/…	59-67 182-191 385-436	0 0 3 1	0.059	4
B100	7_450	54	7_450 3_385 7_345	164-181 510-551	0 0 0	0.004	0
B260	7_450	53	7_450 5_544	164-181	0 4	0.001	4
B81	7_450	52	7_450 1_259	192-205	1 6	0.001	7
108	3_610	51	21_019 7_450	68-181	0 0	0.001	0
B83	7_450	51	7_450 10_019 1_504/…	192-209 437-466	3 2 2	0.201	7
B358	11_424/…	50	5_544 7_450 3_385	68-181 727-752	2 0 0	0.004	2
B269	3_610	48	3_610/… 7_450	68-181	0 0	0.001	0
B281	7_450	43	7_450 1_504/…	238-242	0 4	0.001	4
B268	7_450	41	7_450 1_504/… 7_450	164-181 583-712	0 0 0	0.122	0
B270	7_450	39	7_450 3_385 3_014	267-273 502-505	0 0 0	0.001	0
B282	1_504	36	3_354/… 7_450 3_385	238-242 583-712	1 0 0	0.004	1
B362	1_259	31	11_424 7_450 10_019	389-401 583-624	2 0 0	0.102	2
94	11_456	26	11_456 7_450 11_456	415-422 583-651	0 0 0	0.003	0
B98a	3_496/…	20	10_019 7_345/…	214-263	2 8	0.001	10
622	3_014	19	6_324/… 3_014 3_385	316-388 676-764	0 0 0	0.33	0
B276	1_504	18	3_354/… 3_385 7_450	192-306 521-553	0 0 0	0.021	0
B188	11_424	18	1_259/… 7_450 19_352 7_450	256-265 404-414 583-610	0 0 1 0	0.006	1
B262	3_496	17	4_742 5_544	209-226	0 4	0.001	4
B12	5_544	14	5_544 7_450	583-673	7 0	0.001	7
93	7_450	13	77_450 21_019	583-610	1 0	0.001	1

Table 7.4: A slightly modified version of the table in (Flockerzi, Maydt et al. 2007), distributed under the terms of the Creative Commons Attribution License. The table only shows the explanation with the highest number of recombinations. A detailed explanation of the columns is given in the text.

section 7.2.2.2). The prediction of Recco strongly supports that all HERV-KX sequences are recombinants of two or more loci. The number of differences of the recombinant path to the known loci is usually below six and may be caused by the allelic variation observed in the human genome. The recombination signal is highly significant for all sequences: the first recombination event for each query sequence has a *p*-value below 0.002 (not shown in the table) and a very high Savings value, even if the *p*-value of the path in the table is not significant.

In summary, this study demonstrates that *ex vivo* recombination can complicate sequence analysis considerably. Our findings do not rule out *in vivo* recombination and it is almost impossible to differentiate between *in vivo* and *ex vivo* recombination. However, the hypothesis of *ex vivo* recombination is more likely in our case, as the HERV-KX sequences were not isolated multiple times. The potential for *ex vivo* recombination was also quantified in a controlled experiment. About 10.5% of the sampled gag sequences contained *ex vivo* recombination.

7.2.2 HIV-1 Subtyping

Subtyping is an important task for the analysis of HIV-1 sequence data. The subtype usually provides information on the geographic origin of the virus and allows for tracking the spread of the disease more closely. But the analysis becomes more complicated as the virus recombines and produces recombinant forms. The recombinant subtype structure of these viruses is of particular interest and is often studied in more detail. As a consequence, there are several subtyping datasets that were studied manually in great detail and can be used for validating recombination detection methods (see e.g. (Magiorkinis, Paraskevis et al. 2005)). However, the manual analysis does not guarantee that the results are accurate. In the following, we will therefore compare several methods simultaneously and assume that their consensus provides a good estimate.

7.2.2.1 Materials and Methods

The analysis in this section closely follows the publication of (Magiorkinis, Paraskevis et al. 2005) and compares the result of the publication to the output of Recco and jpHMM, a method for subtyping HIV-1 sequences. The subtypes and accession numbers of the 19 reference sequences are: subtype A (M62320, U51190), B (K03455, M17451), C (U46016, U52953), D (K03454, A34828), F (AF075703, AF005494, AJ249236), G (U88826, AF061641), H (AF190127, AF005496), J (AF082395, AF082394) and K (AJ249239).

Magiorkinis and coworkers analyzed a total of 34 recombinant viral HIV-1 strains. They first constructed a multiple sequence alignment of the query and the reference sequences with Clustal W (Chenna, Sugawara et al. 2003). The recombination analysis was then performed with bootscanning using a sliding window of 400bp and a step size of 50bp. A recombination breakpoint was detected if adjacent regions support a different HIV-1 sub-

type classification with a bootstrap value above 70–75%. The subtype classification was then further confirmed manually by performing a neighbor-joining and a maximum-likelihood phylogenetic analysis for each genomic region. Some sequences or sequence regions could not be assigned to a subtype and were further analyzed with a BLAST search (Altschul, Gish et al. 1990). The result of the BLAST analysis often proposed more exotic HIV-1 variants as parental strains and is not reported in detail by the authors. The following discussion is therefore based on the results of the maximum-likelihood analysis (MLA). MLA usually predicts one of the subtypes for each region (see Figure 1 in (Magiorkinis, Paraskevis et al. 2005)). A special case is that some recombinant sequences cluster to the parents of subtype B and D in selected regions and have two potential parental sequences. This case is represented with an artificial subtype 'P' in the following. As the exact location of the breakpoints are not listed for the MLA, they had to be estimated from Figure 1 in (Magiorkinis, Paraskevis et al. 2005) and are only accurate to about ±200bp.

The MLA may not recover the correct solution, even though it follows a partially hand-driven protocol. A major problem is the hierarchical approach, as errors in the bootscanning step propagate to the phylogenetic analysis and cannot be fixed there easily. If the bootscanning step does not detect a breakpoint, the subsequent phylogenetic analysis is also inaccurate, as it estimates a single subtype for a recombinant region. On the other hand, the bootscanning step may also detect too many breakpoints. The phylogenetic analysis then becomes inaccurate as there may not be enough phylogenetic information in smaller genetic regions. Unfortunately, the publication does not indicate whether the subtyping results were produced with a lot of manual interaction that could partially avoid these problems. In summary, the output of the MLA may be a good indication for the subtype structure, but it is far from accurate.

The analysis for Recco is based on the HIV-1 genome alignment (downloaded on 26.2.2008) of the Los Alamos National Laboratory (http://www.hiv.lanl.gov/) and avoids the alignment step. The whole genome alignment contains all reference sequences and 31 out of 34 recombinant sequences. Three recombinant sequences were not found in the complete genome alignment and are not analyzed in the following: AF076475, X04415, AF005495. All columns in the alignment with a gap in the HXB2 sequence were discarded, so that the numbering of columns corresponds to the HXB2 reference sequence. In a second step, an alignment that contains the query and all 19 reference sequences was extracted for each recombinant sequence, resulting in a total of 31 alignments. Recco was then used to estimate the composition of the query sequence for each alignment and 200 column permutations were performed for p-value computation. As the alignments do not contain many and large gap regions, the gap scoring was not as important as for the analysis of the HERVs. Hence, we simply used a gap penalty of 0.2 for the analysis and checked manually if gaps strongly attracted the α-optimal path.

The subtype composition of the 31 recombinant sequences was also determined using the web-interface of jpHMM (Zhang, Schultz et al. 2006) (see section 4.3.3.4).

7.2.2.2 Results and Discussion

The subtype prediction of all three methods is too unreliable to provide a gold-standard for comparison. The breakpoint positions of the MLA were particularly inaccurate as they had to be reconstructed from a Figure in the publication (Magiorkinis, Paraskevis et al. 2005). In order to allow for a sensible, informative and compact comparison of the results for all three analyses (MLA, Recco and jpHMM), the results were compared and combined manually (see Table 7.5).

As described in the following, the recombination breakpoints were consolidated manually to arrive at a representative, but not too fine-grained segmentation of the genome into regions. The breakpoints predicted by Recco and jpHMM rarely differ much for simple cases. If the two breakpoints fall within 100bp, only the breakpoint predicted by Recco, rounded to 10bp, is provided and the distance to the breakpoint of jpHMM is color coded: green denotes a deviation of ≤30bp, orange 30–50bp and red 50–100bp. If the breakpoint was only predicted by one of the two methods, it is colored black. The approximate breakpoint locations of the MLA then often matched a predicted breakpoint of Recco or jpHMM. If this was not the case, the breakpoint location of the MLA is reported with the prefix '~' as they are very inaccurate. In summary, the breakpoints of Recco and jpHMM are used as the most accurate predictions as the resolution of the MLA breakpoints is very limited.

The breakpoints then define a segmentation of the HIV-1 genome. If there are x breakpoints, the HIV-1 genome is divided into x+1 regions in total and the subtype composition is given for each region and each method. The column "Prediction" shows the output of the MLA, Recco and jpHMM in the first, second and third row, respectively. The output of Recco and jpHMM is color coded to allow for a faster comparison of the results: all differences to the prediction of the MLA are shown in red. A region does not count as a difference if it was predicted as '?' for the MLA or for Recco or jpHMM. Similarly, 'P' matches subtypes 'B' and 'D'. In some cases, a fourth row provides the output of Recco for a lower, but non-significant Savings value. The additional subtype predictions then match some predictions of the other methods more accurately. The "Comments" column also indicates if Recco recovers the exact output of jpHMM or the MLA for a lower Savings value. The MLA sometimes predicts no subtype ('?') for a region if there is no clear signal from the analysis. These regions are not always represented explicitly in the table to avoid generating an excessive number of breakpoints. If Recco and jpHMM do not predict a breakpoint at the boundaries of a '?' region and the region is rather small, the predicted subtypes of the MLA in the adjacent regions are simply extended to cover the unclear region. JpHMM can also directly predict the circulating recombinant form '1', a recombinant between subtypes A and E.

Table 7.5 shows the output of the methods. Sequence 1 is a very simple example for a subtype prediction. All methods predict the subtype composition ADA and both breakpoints 3060 and 4010 were predicted by jpHMM and Recco with a deviation of less than 30bp from each other. Hence, the methods predict subtype A for positions 1 to 3060, subtype B for positions 3061 to 4010, and subtype A for the rest of the sequence from positions 4011 to 9719. Sequence 2 presents a more complex example. Recco only predicts one recombination event with a Savings value of 186 and a highly significant *p*-value. The next path with an additional recombination event corresponds to a lower Savings value of 19 and does not have a significant *p*-value anymore, but recovers the solution of jpHMM. It is unclear whether the MLA solution is correct, as it is the only method that predicts subtype C for a small region between positions ~6800 and ~7100. If the Savings value is reduced to 8, Recco actually also detects subtype C between positions 6790 and 6930. However, Recco is very sensitive to this Savings value and also finds a (potentially correct) intra-subtype recombination in subtype C at position 6100 and a (probably incorrect) recombination close to the end of the alignment due to gaps. Sequence 2 also shows that some recombination events may become obsolete for lower Savings values (see section 5.6.2 for details). Recco predicts a recombination breakpoint at position 6320 with a Savings of 186 between the parental sequences U52953 (subtype C) and U51190 (subtype A). If Savings is reduced to 8, the breakpoint shifts to 6330 and the parental sequences also differ (U46016 and M62320), but are still of the same subtype. Fortunately, the difference does not matter for the purpose of subtyping in this case.

The predictions of the three methods often disagree with respect to the number of segments and recombination breakpoints. Sequence 4 nicely illustrates these characteristic differences. Recco predicts the least amount of recombination and only infer subtype A for the region between position 6340 and 8320. MLA infers an additional region of subtype A between position 3790 and 4050. Finally, jpHMM predicts the most recombination breakpoints and also finds subtype A between positions 5790 and 5950. Recco performs particularly well in this case, as it recovers the MLA solution for Savings=10 and the jpHMM solution for Savings=7.5. However, the solution for Savings=7.5 is already very susceptible to noise and predicts subtype A only up to position 5900 and not up to position 5950 as jpHMM. Subtype A actually only incurs ten mutations less than subtype D in the region predicted by jpHMM, and the question is therefore whether ten mutations suffice to justify two recombination events.

In general, jpHMM is very aggressive and tends to predict many recombination breakpoints. Regions that are classified as unclear by the MLA often show multiple recombination breakpoints in jpHMM (see sequences 16, 24 and 30). But other regions are also often subject to a higher number of recombination breakpoints in jpHMM (for extreme examples see sequences 6, 7 and 9). Recco is very conservative and predicts fewer recombination breakpoints than jpHMM and the MLA. One exception is due to a potentially wrong result for the MLA (see sequence 3). All other exceptions occur for alignments

ID	Accession Number	Prediction (MLA, Recco, jpHMM)	Breakpoint Positions (1-9719)	Comments
1	AF071473	ADA ADA ADA	3060 4010	
2	AF067156	CACA? CAAAA CAAAC	6320 ~6800 ~7100 9130	Recco recovers the jpHMM result with Savings=19 (p-val=0.35). The C at the 3rd position of the MLA may be noise.
3	AF071474	CCACA ACACA ACACA	1190 2010 5175 6910	The MLA may be insensitive for small stretches of differing subtypes and not detect A at the beginning.
4	AF075701	DADPPPAD DDDDDDAD DADDADAD	3790 4050 ~5300 5790 5950 6340 8320	Recco recovers MLA result with Savings=10 (p-val=0.2), and the jpHMM result with Savings=7.5.
5	AF075702	AAPAAACCC DADAAADCA ?ADADADC?	790 1720 2800 6860 7320 8500 8820 9410	Gaps at the end of the alignment may negatively affect the solution of Recco.
6	AF076474	AGHHG?AA???KKKKK?? AHHHHHHHHHKKKKHHHH AGHCGHAKHGAKAFHGHK	1990 2890 3850 4380 4950 5370 5810 5950 6380 6480 7030 7680 7900 8270 8660 8830 8940	JpHMM probably introduces too many breakpoints between positions 7680 and 8660.
7	AF192135	AJ??JA?JJJJ?JJ AJJJJJJJJJJJJ AJDCGACJGJCAJC	2010 2580 2870 3290 3620 4080 4990 6420 6800 7170 7690 8310 8980	A typical case where jpHMM detects (too) many recombination breakpoints.
8	AJ237565	AAKA?H BAKAAH AACACH	820 2750 4130 5700 5800	Recco (erroneously) predicts B as the subtype for the first region because of a gap.
9	AJ276595	ACGCAGGAGCCCGGGG AAGGGGGGGCCCGGGG ACGCACGAACACGAGA ACGGGGGGGCCCGGGG*	1530 2190 3180 3720 4000 4200 4950 ~5300 5590 7030 7150 8570 8750 9150 9360	Recco generates the output (*) for Savings=22 (p-val=0.1).
10	AJ276596	AGAGGGG AAAGGGG AGAGAGA	2220 3210 4200 8730 9090 9360	Recco recovers the MLA result with Savings=20 (p-val=0.45).
11	U76035	GJGJJGAG GJJJJGAA GCGFCGAG GJJJJGAG*	3520 4140 4990 5720 5930 6320 8370	Recco generates the output (*) for Savings=22 (p-val=0.065).
12	U86780	AAAC?ACAAA AAAAAACAAA ACACCACACA	1960 2290 3210 3800 ~5000 6240 6590 7320 7510	
13	U88823	CACACAC CCCCCAC CACCCAC	1820 2700 ~4700 ~5000 6040 8300	Recco recovers the MLA result with Savings=25 (p-val=0.1).
14	U88825	GAG GGG GAG	4240 5920	Recco recovers the MLA result with Savings=11 (p-val=0.92).
15	AJ239083	AGAGAO AAAAAD AGAGAA	2240 3210 4210 4870 5830	The last region refers to subtype O (other) and interferes with Recco's analysis, as it introduces many mutations and gaps.
16	AJ404325	GGHJGJJKJ?????A?G? GGGGGJJJJAAAAAGGG HGHCGGKKBAA1H1AAGH	1060 2140 2700 4130 4880 5100 5500 6050 6280 6640 6910 7190 7430 8330 8440 8640 9010	
17	AF362994	BA???B BGCKBB B1111B	6200 ~6500 7410 7480 7700	Gaps cause Recco to erroneously detect the third and fourth breakpoint. The subtypes C and K are probably incorrect.

Table 7.5: The table lists the HIV-1 subtype prediction for the MLA, Recco and jpHMM. See text for details.

ID	Accession Number	Prediction (MLA, Recco, jpHMM)	Breakpoint Positions (1-9719)	Comments
18	AF377957	AGAAGAGAGGG AAAGGAGGGGG AGAAGAAAAGA	2220 3240 5050 5280 6350 8200 ~8500 ~9000 9220 9350	
19	AF377959	AGGFFA?GGAKH?HHA AKKKKKKKKKHHHAA AAGFGACGAAKHGHHA	1990 2110 2500 2970 3490 3750 4440 4670 ~4900 5600 6100 6360 6900 7320 7620	The predictions disagree up to position 5600. It would be interesting to analyze this sequence in more detail.
20	AY046058	AGGAKAGJJJAAA AKKKKJJJJJJAA AAGAKAGAGBJAAH	1990 2170 2430 3090 3920 4070 4780 6050 6170 6260 6340 6510 9260	JpHMM and the MLA roughly agree up to position 4780, while Recco and the MLA concord for the rest.
21	U51188 CRF01	A?GA AAAA 1111	~6500 ~7800 ~8500	JpHMM contains the sequence to subtype (CRF01) in the training set and correctly detects it.
22	AF063224 CRF02	AGAGAGA?GAG AAAAAAAGGGG AGAGAGAAGAG AGAGGGAGGGG*	2240 3210 4210 4920 6070 6190 8160 8370 8560 9150	Recco generates the output (*) for Savings=15 (p-val=0.8). The MLA does not introduce an unknown region, but the breakpoint could not be determined accurately enough and is marked by '?'.
23	AF193277 CRF03	ABA ABA ABA	2700 8670	
24	AF119819 CRF04	AG??FK?GA?AA???KAH? AAAAAAAAAAAAKKKKKHH AGCAFFFAAJAGGACKAHC AFFFFFFFAAAAKKKKKHH*	1890 2250 2530 2880 ~3000 3720 4580 4800 5460 5690 6260 6660 6830 7180 7510 7790 8310 8660	Recco generates the output (*) for Savings=31 (p-val=0.1).
25	AF076998 CRF05	DFDFDFDF DDFDFDFF DFDFDFDF	2800 3360 3940 5240 6040 8360 8710	Recco recovers the MLA solution for Savings=12 (p-val=0.6).
26	AJ288981 CRF06	AGGGKK?GGGJJJG?JJJJJ AGGGKKKKKJJJGJJJJJJ AGACCKDDAAAJKGGJGJCA AGGGKKKGGGJJJGJJJJJJ*	1430 2460 2760 3150 3480 3840 4060 4210 4370 4880 5100 5640 6030 8170 8330 8670 8890 8990 9280	Recco generates the output (*) for Savings=19 (p-val=0.06). The output is similar to the MLA solution and assigns subtypes to all regions.
27	AF286230 CRF07	CBCBCCPCCPC CCCCCBBBCCC CBCBCCBCCBC	2060 2550 2980 3180 5700 5850 6430 6650 8840 9060	Recco is more conservative than the other methods. It would be interesting to study this case in more detail.
28	AY008717 CRF08	CBCBCCC CCCCCBC CCCBCBC	~1200 ~1500 2850 3170 8800 9000	Recco recovers the jpHMM solution for Savings=11 (p-val=0.1).
29	AF289548 CRF10	DCPCPCDCDC DDDDCDDDC DCDCDCDDDC	3220 3630 4650 5390 5790 6400 ~6800 ~7000 8870	
30	AF179368 CRF11	AJKJJJAJJGAJG??? AAAAAAAAAAAJJJJJ AJAAGCAKJGAAGJA1 AJJJJJJJJGAJJJJJ*	2130 2480 ~2800 3170 3580 4170 5830 6150 6370 6820 7610 7970 8570 8750 8950	Recco generates the output (*) for Savings=25 (p-val=0.06).
31	AF385936 CRF12	KFBFPFBF BFBFFFFF BFBFBFBF	950 2990 3710 5950 6240 8480 8630	Recco recovers the jpHMM solution for Savings=12 (p-val=0.25).

Table 7.5 (continued).

with larger gap regions: Recco erroneously infers recombination breakpoints because of a gap for sequences 5, 8 and 17. The disparity between Recco and jpHMM regarding the number of breakpoints probably originates from different underlying principles. Recco is based on a hypothesis test and only accepts a recombination event if it incurs a significant

Savings value. In other words, Recco tries to limits the number of falsely detected recombination events. In contrast, jpHMM tries to find the subtyping result with the maximum likelihood in the hidden Markov model and does not penalize additional recombination events as much.
The predictions of the three methods agree for many parts and many sequences of the alignment. The MLA is often similar to Recco or jpHMM, and the consensus may present an even more accurate prediction. Nevertheless, some sequences show a very complex recombination signal, such as sequences 6, 19, 20, 24 and 30. These sequences are very difficult to analyze, as the output of all three methods disagrees. It would be interesting to analyze these sequences in more detail and estimate their recombination structure accurately. Unfortunately, a manual analysis of recombinant sequences is usually very time consuming and is out of the scope for this validation.
In summary, Recco performs comparably to jpHMM, even though it was not designed for subtyping. Recco is much more conservative and only infers recombination if there is substantial evidence for it. The Savings value as a measure of robustness is also very useful in practice, as it allows exploring different recombination hypotheses quickly. However, Recco does not model gaps very well, such that the analysis is sometimes negatively affected by gaps in the alignment. Extending Recco to a full subtyping method also requires a more detailed analysis of the robustness. Recco and jpHMM only infer a subtype for every position in the alignment. A measure of robustness for the subtype prediction at each position would be extremely useful.

7.3 Conclusion

Recco works very well for detecting recombination in an alignment and finding the structure or composition of potentially recombinant sequences. Recco's power of detecting recombination in alignments is particularly encouraging. Recco is also very intuitive and easy to use – at least in our subjective view. Many other popular or graphical methods for analyzing recombination do not directly provide a recombination hypothesis for the detected recombination signal (for example Topali, RDP and Geneconv) or are too insensitive (for example Bootscanning or Simplot). However, Recco has several shortcomings that complicate the analysis of biological datasets. The analysis of Recco can be adversely influenced by gaps in the alignment, just as for many other methods. The analysis based on the Savings feature also fails if there are sequences similar to the recombinant in the alignment.
Savings works well for detecting the recombinant sequence in an alignment and is particularly effective for reconstructing the potentially recombinant composition of a single query sequence. Savings is also rather easy to interpret and therefore particularly useful in a manual analysis of the alignment. However, Savings cannot recover the recombination signal if there is a sequence similar to the recombinant in the alignment. An automated strategy for consolidating the output of Recco for different query sequences would

therefore be valuable in practice and may even supersede OptSavings for recombination detection. The next chapter presents some ideas for future work on such a strategy.

While Savings clearly performs better than Geneconv for specific recombination scenarios, it only performs comparably to Geneconv on random ARGs. OptSavings was developed as a simple strategy of avoiding the limitation of Savings and of investigating the cause of the reduced performance for random ARGs in more detail. The power of OptSavings in detecting recombination is impressive, particularly for low recombination rates. OptSavings is better than all other methods used for comparison for most parameters θ and ρ. However, OptSavings is not as suitable for manual analysis of the alignment, as it does not allow for a simple interpretation like Savings. We also cannot use OptSavings to estimate the composition of the query sequence regarding a set of reference sequences, as OptSavings may remove some of the reference sequences and produce incorrect results. In summary, OptSavings should only be regarded as a first step towards a more comprehensive analysis of the recombination signal in an alignment.

Recco also performs well for real alignments and in particular for estimating the composition of a query sequence in the alignment. In comparison to jpHMM, Recco is very conservative and only predicts few recombination events. Such a conservative approach is preferable if we want to confirm that recombination took place and if every estimated recombination event has serious implications for the interpretation of the analysis. However, it is more appropriate in some cases to infer a composition that does not penalize recombination as strongly as Recco. Consequently, it may be interesting to devise a scheme for finding a less conservative threshold for the Savings value. Still, the most serious limitation of Recco is the scoring of gaps. Just like many other methods, Recco may infer a wrong output if there are many or large gap regions in the alignment. It is time consuming to scan manually for the effects of an unsuitable gap penalty. Even worse, manually constructing the correct solution is often difficult or even impossible. A particularly relevant task for the future is therefore to find a robust and appropriate method for scoring gaps.

8 Outlook

The validation showed that Recco works well on simulated and real sequence data. Nevertheless, there are many aspects that could still be improved in Recco and some of them were already discussed in previous chapters. The most pressing need for analyzing real sequence data is an approach for scoring gaps more appropriately and the corresponding problem has been studied in more detail in Chapter 6. Section 8.1 discusses several aspects about the problem of recombination analysis in general and is not specific to Recco. Section 8.2 then turns to problems specific to Recco and discusses their potential resolution.

8.1 Fundamental Aspects of Recombination Analysis

Chapter 3 and 4 describe the ARG and define recombination analysis and the questions it addresses for the matter of this thesis. In reference to this, the following section focuses on some simple aspects of recombination analysis and discusses several extensions to the problem of recombination analysis as we have defined it.

Nevertheless, there are more complex aspects as well. For example, recombination analysis is limited to some population genetic scenarios and only addresses questions about the ancestry of the sequences. If the sequences are generated with a recombination rate that is sufficiently high compared to the mutation rate, the ARG cannot be estimated anymore from sequence data and each site evolves according to a different tree. Consequently, the ARG and the questions proposed for recombination analysis are not of great interest anymore. Instead, we may assume that sites are almost in complete linkage equilibrium and rather utilize this information in our analysis, ignoring the difficult problem of estimating the ARG. However, it is unclear for HIV-1 whether the observed recombination rate inside a patient is high enough to justify such an approach.

8.1.1 Timed Sequence Data and Other Additional Information

Starting with a multiple sequence alignment, we have structured the analysis of recombination into five questions that are usually addressed sequentially (see section 4.1). However, we often have additional information about the sequences in the alignment that impact the questions we would like to address. For example, the sampling time of sequences can restrict admissible phylogenetic relationships in standard phylogenetic analysis and allows for a more accurate estimate. This approach generates new insights in influenza evolution and nicely illustrates the impact of modeling additional time information (McHardy, in preparation). The situation is similar for recombination analysis, as we can often restrict the hypothesis space and arrive at more accurate estimates. We can sometimes exclude potential recombination events if the sequences were sampled at different points in time, as it is much more likely that older sequences are parental to newer ones than vice versa. Geographic location and other transmission barriers also provide valua-

ble information, as recombination is more likely to occur between sequences that are sampled at the same location. For example, HIV-1 only produces recombinant sequences if a cell is infected with two or more strains and we can assume that sequences sampled from different individuals do not recombine, at least not as vigorously as sequences sampled from the same individual. In general, it is the responsibility of the investigator to understand the additional information and restrictions in detail and to carefully interpret the results of the recombination analysis.

Nevertheless, some of this information can be incorporated explicitly into a method. HIV-1 subtyping is a prominent example of a tailored problem definition, as it assumes that all sampled sequences are recombinants of the known subtypes. In other words, HIV-1 subtyping assumes that recombination events between recently evolved and potentially recombinant sequences are not relevant for the analysis. Information on the sampling time of sequences is even more interesting, as older sequences are more similar to the parental sequences of a recombinant and can therefore increase the power of detecting older recombination events considerably. Surprisingly, current methods for recombination analysis and for inferring population genetic parameters such as ρ and θ do not use the sampling time of the sequences. A simple approach to incorporating such information into Recco is to restrict the set of possible parental sequences in the alignment. For example, we can remove all sequences with a newer time stamp than the query from the input alignment and reconstruct a recombination hypothesis only using parental sequences that are older than the query.

8.1.2 Validation and the Strength of the Recombination Signal

The goal of recombination analysis as we have defined it in this thesis is to infer information on the recombination events in the ARG underlying the sequence alignment. The strategy for validation is usually to simulate sequences along an ARG and to investigate if a method for recombination analysis recovers the recombination events from the sequence data. Ideally, we would like to evaluate the power of detecting the recombinant sequence or the breakpoint for ARGs drawn randomly from the coalescent and not to rely on predefined ARGs as in chapter 7. But there is a serious complication for this strategy: the ARG is not unique and many different ARGs generate very similar distributions of sequences. In other words, recombination events may have no or only little impact on the sequence data and cannot be reconstructed because of the structure of the ARG and not because of a deficit in the method.

A first step into this direction is to classify the recombination events regarding the changes of local tree topology that they cause (see section 3.3.3.4) and determine the power of detecting recombination events for each type in random ARGs (Wiuf, Christensen et al. 2001). But the strength of the recombination signal also depends on other topological features of the ARG and hence, we can ask for a quantitative measure of the signal strength: given an ARG, how strong is the signal regarding some recombination

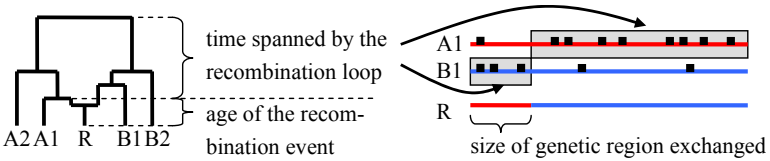


Figure 8.1: Shown are the factors that influence the recombination signal in our example ARG. The left panel shows the ARG and the right panel presents an exemplary alignment containing the two parental sequences and the recombinant. Mutations regarding the recombinant sequence are visualized as black dots and the parts of A1 and B1 not parental to the recombinant are marked by a grey background. The time spanned by the recombination loop determines the dissimilarity of the parental sequences and therefore only influences the number of mutations in the gray region. The age of the recombination event also increases the number of mutations in the gray region, but affects the other genetic regions of A1 and B1 similarly. Finally, the location of the recombination breakpoint determines the size of the exchanged genetic region and therefore also affects the number of exchanged mutations.

event? To make the analysis simpler, we can assume that the ARG only contains a single recombination event. In this case, we could measure the strength of the recombination signal in a generated sequence alignment by counting the number of mutations that are exchanged by the recombination event, a measure similar to the Savings value. But we can also quantify the recombination signal in an ARG independent of the realization of the mutation process and only work with the branch lengths of the ARG. A strong recombination signal then requires branch lengths that do not fit well to any possible evolutionary tree.

Intuitively, it is clear that there are three major factors that determine the strength of the recombination signal: the age of the recombination event, the time spanned by the recombination loop and the size of the genetic region that is exchanged (see Figure 8.1). These factors require that we normalize the ARG and move all non-merging events and particularly the recombination event as far as possible into the past. The age of the recombination event is directly related to the number of mutations that the recombinant sequence accumulated. Consequently, older recombination events only exhibit a weak and degraded recombination signal, as the similarity to the parental sequences is decreased. The time that the recombination loop spans determines the similarity of the parental sequences that recombine. Larger recombination loops lead to stronger recombination signals. The size of the genetic region that is exchanged depends on the location of the recombination breakpoint. If the breakpoint is at the boundary of the alignment, fewer mutations are exchanged and the recombination signal is weaker.

Figure 8.2: Other factors also influence the recombination signal and therefore determine our ability to detect the recombination event in the alignment. **Left:** We can estimate the ancestral sequences that participated in the recombination more accurately (marked by a circle) if the sequences A1 and B1 did not diverge as much from the ancestral sequences. **Right:** Likewise, we can reconstruct the ancestral sequences more accurately if the alignment contains more than one descendant of each ancestral sequence.

Interestingly, these three factors also support the statement that Recco is a good approach. The Savings value that we expect for a recombination event depends on the recombination detour, which in turn is defined by the three factors introduced above – at least if there is no sequence similar or equal to the recombinant in the alignment. In other words, Recco's ability to recover recombination events depends on their Savings value, which is similar to the signal strength of the recombination event in the ARG.

A quantitative measure of the recombination signal is also of direct practical use. We could evaluate the power of detecting a recombination event depending on the strength of the signal and quantify more accurately the sensitivity of a method regarding weak and strong recombination signals.

Finally, there are several other factors that determine the strength of the recombination signal to a lesser extent. For example, the mutation model influences the effect of recurrent mutations. If the mutation model has only few states, recurrent mutations are more likely and the recombination signal is more easily eroded by recurrent mutations. Another factor is the information contained in the alignment regarding the ancestral sequences that participated in the recombination event. The recombination signal is easier to detect if sequence A1 is more similar to the ancestral sequence that recombined (see Figure 8.2).

8.2 New Developments for Recco

This section discusses several developments that have a direct impact on Recco. There are many other simple and incremental steps that could also improve the estimate of Recco for biological sequence data, but that are not discussed in detail. For example, Recco only uses Hamming distances for the mutation cost and currently does not estimate a mutation cost matrix from the sequence alignment or account for mutation hotspots. A more accurate model for the mutation cost could improve the accuracy of Recco, but also makes the interpretation of the Savings value more difficult. The Savings value then does

not anymore represent the number of mutations eliminated by an additional recombination event and is less intuitive.

8.2.1 The Distribution of the Savings Value

Recco depends critically on X or X_{i^*}, the distribution of the Savings value for column permutations of the alignment. The distribution is estimated by a compute intense permutation approach in the current implementation. However, the plots in section 5.6.3.2 suggest that the distribution of the Savings value is approximately Poisson-distributed. The information on the type of the distribution would be extremely valuable as it allows for estimating the distribution of the Savings value with a parametric approach. Given that the Savings value is Poisson-distributed, we only have to estimate the parameter of the distribution and can then estimate the quantiles of the distribution much more accurately from very few column permutations. The benefit is twofold: the approach is much faster and it can also estimate smaller p-values.

It may even be possible to estimate the parameters of the Savings distribution entirely by analytical means. Clearly, the distribution of the Savings value is completely defined by the distribution of column states in the alignment. Consequently, it may be possible to devise an efficient algorithm that estimates the distribution of the Savings values directly, for example by counting the number of column permutations corresponding to a certain Savings value. We have also introduced a simple bound σ_i for the Savings value of query sequence i. The bound σ_i is interesting as it correlates well to the mean of the Savings distribution and describes an important aspect of the distribution solely by analytical means. A closer study of σ_i may guide the way for an entirely theoretical derivation of the Savings distribution.

8.2.2 Subtyping and the Robustness of α-Optimal Paths

The analysis of new sequences regarding their subtype composition is important for HIV-1 and for other pathogens. Even though Recco was not designed for this task, it performs well compared to other methods (see section 7.2.2). But it is also interesting to complement the basic subtyping analysis by additional steps in the procedure and assess the robustness of the estimate. Such an extended analysis is not offered by any current subtyping method so far and is also interesting for the output of Recco in general. The α-optimal path of Recco only is a point estimate of the subtype composition of the query sequence, just like the output of jpHMM and other subtyping methods. The subtype prediction can differ from the correct solution by three aspects: the number of segments (or recombination breakpoints), the location of the breakpoints and the predicted parental sequences. We can address these aspects at least partially with the sensitivity analyses presented in section 5.4.

The parametric analysis determines if the extracted path is optimal for a large interval of Savings values and hence, whether it is robust to small changes in α, to the cost of recombination relative to mutation or to noise in the mutation process. Fortunately, paths

corresponding to a higher or lower Savings value often change the prediction locally and only add or remove a recombination breakpoint, but do not affect the rest of the prediction. Consequently, we can even determine whether a particular recombination breakpoint or parental sequence is stable for a larger interval of Savings values. However, the interpretation is not always as straightforward, as the parametric analysis can also cause more complex structural changes of the α-optimal path (see section 5.6.2). Alternatively, we can keep α fixed and assess the robustness of the prediction regarding small deviations from cost-optimality using the values c_{ip} and r_{ip} described in section 5.4.1. This type of analysis can assess whether the prediction of a parental strain in a region is particularly stable or if there are other parental strains that also fits well.
As shown above, Recco already implements some tools for analyzing the robustness of a prediction. These analyses allow the user to investigate the robustness manually and could provide the building blocks for a fully automated analysis.

8.2.3 The Structure of α-Optimal Paths as α Changes

The recombination event list provides an intuitive summary of all potential recombination events for the query sequence. But it is sometimes difficult to interpret the list, as recombination events corresponding to a small α-value may become obsolete as the α-value increases – or as the Savings value decreases (see section 5.6.2). It would be very interesting to study this effect in more detail and understand the conditions when it occurs and how the α-optimal path changes. For example, several recombination events in section 7.2.2 are only predicted transiently in the α-optimal path as the α-value is increased or as the Savings value is decreased. All of these recombination events follow the same pattern: if a recombination event becomes obsolete, at least one of the two parental sequences surrounding the respective recombination breakpoint also change. To be more specific, if a recombination event becomes obsolete for a larger α-value, it is replaced with another recombination event that chooses different parental sequences and that introduces a recombination breakpoint at a different position. However, the new breakpoint position is usually close to the former breakpoint position and the difference is often negligible. Studying the optimization problem and the structure of the α-optimal path from a mathematical point of view should allow for understanding the differences of the α-optimal paths as α changes. As an alternative, we could also implement an approach to reconstruct the ARG as described in section 8.2.5. The problem should then occur less frequently as the parental sequences are determined more accurately.

8.2.4 Improving the Accuracy of Recco

Recombination analysis not only detects recombination in the alignment, but also tries to find the recombinant sequences and the recombination breakpoints. Validation shows that the latter tasks are quite challenging and that Recco does not always perform better than other methods. There is probably a simple reason for this shortcoming: Recco also detects recombination events if the query sequence is a sequence that is closely related or

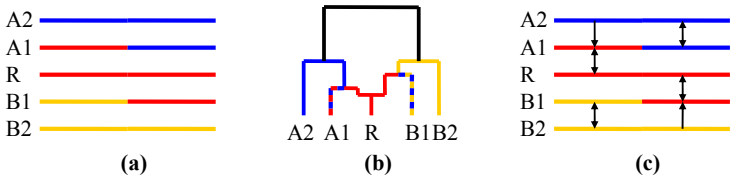


Figure 8.3: The colors in both plots indicate homology. **(a)** The related sequences A1 and B1 can also be described as a recombinant which makes it difficult to discriminate them from the true recombinant R. **(b)** The ARG visualizes homology and the predictions of Recco as in (a). **(c)** We can also annotate the predictions of Recco for every sequence and every region of the alignment using a homology graph, indicated by the arrows in the plot. For example, the arrows from A2 to A1 show that A2 is derived from A1 over the full length of the alignment.

parental to the recombinant. For example, sequence A1 in Figure 8.3 is closely related to the recombinant sequence R1 and is often (and incorrectly) detected as a recombinant sequence by Recco. However, the Savings value for related sequences is usually not as large as the Savings value for the recombinant sequences and may correctly discriminate between recombinant and closely related sequences. Finding the recombinant sequence remains difficult, though. There is another problem connected to detecting related sequences as recombinants: the recombination signal is usually weak for these sequences. As a consequence, the predicted recombination breakpoint is usually quite inaccurate for sequences such as A1. In summary, some recombination events are significant, but inaccurate and only refer to sequences related to or parental to the recombinant sequence.

We can address these shortcomings at least partially by post-processing and consolidating the output of Recco for different query sequences. For simplicity, we only discuss the example alignment in the following and assume that the alignment results in an output in concordance with the ARG analysis (see Figure 8.3 (a)). In this case, Recco predicts sequence R as a recombinant of A1 and B1, sequence A1 as a recombinant of R and A2, and sequence B1 as a recombinant of B2 and R. Recco therefore predicts that the left part of R was derived from A1, and that the same part of A1 came from R. We can now identify A1 as a closely related sequence, as the right part of A1 comes from A2 and incurs more mutation cost than the right part of R. Hence, we should only keep the more accurate recombination event predicted for R and discard the recombination event for A1. The same argument identifies B1 as a closely related sequence.

We may even do better if we optimize the recombination breakpoint location for all involved sequences R, A1 and B1 simultaneously. We could find the breakpoint location that minimizes the cost required for deriving A1 from R and A2, R from A1 and B1, and B1 from B2 and R. Alternatively, we can process the output of Recco for all sequences simultaneously (see Figure 8.3 (c)). We can represent the output of Recco for a site in the

alignment and all sequences as a directed homology graph, where nodes correspond to sequences and edges indicate homology. The homology graph has one outgoing edge for every node and should not contain cycles involving three or more nodes, as these cycles are contradictory to any possible local coalescent tree. Then, we can use an approach similar to the recombination parsimony problem (see section 4.2.7) where each node corresponds to a homology graph and each homology graph incurs some mutation cost for some position in the alignment. We can then find the cost optimal position for a transition from one homology graph to another, thereby optimizing the location of the predicted recombination breakpoints. In principle, we could also use this approach without restricting the set of homology graphs in advance. The output then does not provide the optimal coalescent tree for each position in the alignment, but only presents a restricted history for the set of sequences that often does not reach far into the past (see the colored parts in Figure 8.2 (b)). However, the number of homology graphs probably also grows exponentially with the number of sequences and the approach would not result in a significant speedup compared to the approach for the full recombination parsimony problem.

8.2.5 Reconstructing ARGs With Recco

The validation shows that OptSavings performs much better than Savings for detecting recombination in sequence alignments corresponding to random ARGs (see section 7.1.3) and suggests that Recco is quite limited as it can only detect recent recombination events. But OptSavings does not separate recombinant from non-recombinant sequences as well as Savings and also does not allow for a simple interpretation like Savings. Ideally, we would like to have an approach that combines the excellent power of OptSavings and the interpretability of Savings. A promising approach is to reconstruct the ARG bottom up from the output of Recco. The most important insight is that the output of Recco recovers part of the ARG underlying the alignment, at least if the mutation cost in Recco measures distances in the ARG accurately enough and if the parameter α is set appropriately (see section 5.5). The heuristic algorithm could then proceed as follows (see Figure 8.4):

i) **Determine the composition of all sequences in the alignment, taking each sequence as a query in turn.** As for subtyping, we can choose an appropriate α-value by finding the significance threshold for α regarding column permutation and then use the α-optimal path as an estimate for the composition of the query sequence. This permutation approach is computationally expensive, as we have to process many column permutations for each query sequence and each step of the algorithm. As an alternative, we could also choose a fixed α-value and use it throughout the algorithm. The problem is that a fixed α-value does not adapt the smoothing parameter α to the query sequence and the specific sequence background of the query sequence and is therefore inferior to determining α by column permutations. For simplicity, we assume in the following that the composition of all query sequences is correctly estimated and that there is only one α-optimal path for each query sequence.

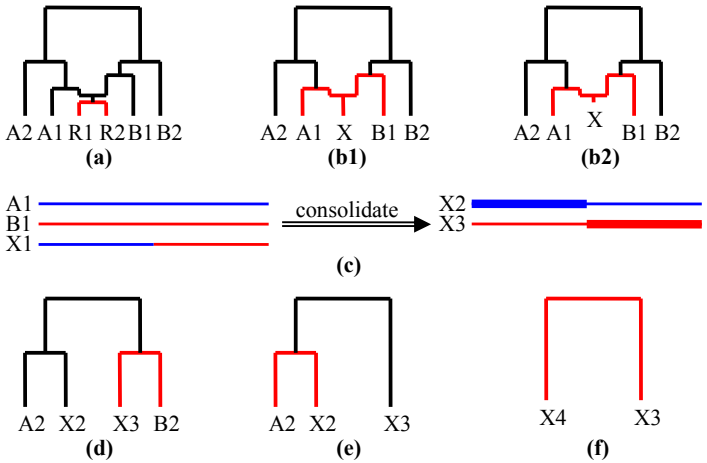


Figure 8.4: The example ARG is reconstructed bottom up with the algorithm described in the text. Each ARG shows an iteration of the algorithm. The α-optimal path with minimal cost is shown in red; sequences arising from consolidating the path are numbered X1 to X4. **(a)** The α-optimal path that incurs the smallest mutation cost explains R1 with R2 (or equivalently R2 with R1). **(b1)** Consolidating R1 with R2 creates a virtual sequence X1 sampled at the current time point. **(b2)** If we consolidated R1 and R2 by computing the minimum cost we would get a virtual sequence X1 that is ancestral to R1 and R2. **(c)** Consolidating sequence X1 requires to incorporate it into the respective parts of A1 and B1, leading to the new sequences X2 and X3. Bold parts of the sequences are the consolidated result. **(d), (e), (f)** The next steps simply construct the remaining tree bottom up using an approach similar to UPGMA.

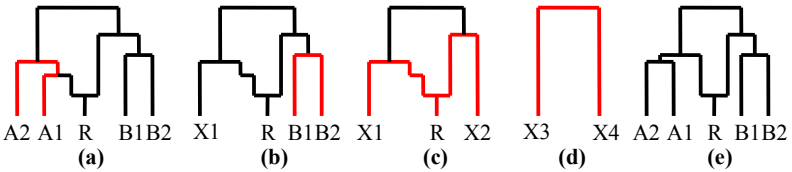


Figure 8.5: The algorithm fails if the segments of the recombinant sequence coalesce at very dissimilar time points. **(a)** Step ii) does not and cannot choose the correct sequence to consolidate. Neither consolidating sequence A1 (or A2) nor R results in the correct ARG topology, and consolidating B1 and B2 incurs a higher cost. **(b), (c), (d)** The other steps of the algorithm build upon the wrong decision in step (a). **(e)** As a consequence, the inferred ARG is incorrect.

ii) **Find sequence *i* corresponding to the *α*-optimal path with the lowest mutation cost.** The reasoning for this step is that the query sequence with the lowest mutation cost is the first to coalesce to other sequences going back in time and that we want to minimize the overall cost of the ARG. The recombination cost is irrelevant if α is chosen appropriately, as only the mutation cost between two sequences measures the time of divergence. A potential source of error is that different segments of an α-optimal path with recombination may coalesce at different time points and cause inconsistencies in the time of coalescence inferred (see below for details).

iii) **Consolidate the *α*-optimal path of sequence *i* and remove sequence *i* from the alignment.** The α-optimal path for sequence i stores information on the most recent coalescent events. We can encode and fix these coalescent events by consolidating sequence i with the predicted parental sequences and then removing sequence i from the alignment. Now suppose that the α-optimal path for sequence i predicts that sequence j is most similar for all positions of the sequence and that we would like to consolidate this coalescent event. A simple approach then allows for multi-sets of nucleotides at each sequence position and consolidates sequence i by computing the union of the multi-sets. To be more specific, consolidating two sequences **u** and **v** with sequence states u_p and v_p then results in a sequence **w** with $w_p=u_p \cup v_p$. For example, consolidating states {A} and {A,G} would result in a state {A,A,G}. There are two options for defining the mutation cost between two multi-sets. The most appropriate choice for a parsimony-based approach like Recco is to define the mutation cost between two multi-sets a and b as the average mutation cost between all states contained within a and b:

$$m(a,b) = \sum_{x \in a} \sum_{y \in b} \frac{m(x,y)}{|a||b|}. \tag{8.1}$$

In other words, if **w** is the result from consolidating **u** and **v**, we compute the mutation cost from sequence **t** to **w** as the average of the mutation cost between **t** and **u** and the mutation cost between **t** and **v**. The mutation cost between consolidated sequences therefore still measures distances in the ARG referring to sampling times at the bottom of the ARG, as shown in Figure 8.4. Alternatively, we could also define the mutation cost between two multi-sets a and b as the minimum over all costs $m(x,y)$. Consolidating two sequences then estimates the ancestral sequence in a parsimony sense and the mutation cost would measure distances in the ARG referring to different time points (see Figure 8.4 (b2)). The first approach is preferable, as the latter can lead to inconsistencies in the reconstructed ARG.

iv) **Repeat steps i) to iii) until there is only one sequence left in the alignment.** Each iteration chooses one sequence of the alignment, determines the composition of the sequence regarding the other sequences, consolidates the sequence with the other

sequences and then removes the sequence from the alignment. If there is only one sequence left, we have reconstructed all coalescent events in the ARG.

The proposed algorithm is actually equal to UPGMA for constructing phylogenetic trees if it is used on a sequence alignment without recombination. Each step in the iteration consolidates the two sequences that are closest to each other and then averages the mutation cost of the two sequences to all other sequences. The proposed algorithm also addresses the same problems as OptSavings and can recover older recombination events that are hidden by duplicate or similar sequences. But it probably also detects the recombinant sequences more accurately than OptSavings, as the algorithm does not optimize costs over different sequence subsets and is therefore less susceptible to noise.
The algorithm only describes a greedy and heuristic approach for reconstructing ARGs and makes several strong assumptions. Step i) assumes that the estimated composition of all query sequences is accurate or accurate enough. Errors can occur simply because the mutation process is stochastic or because we choose the wrong value for α and recover a path with the wrong number of recombination events. Consequently, it is particularly interesting to combine step i) with an approach that enhances the accuracy of the estimate as described in section 8.2.4. But step ii) and iii) can introduce errors even if the composition of the query sequences is accurately estimated, as the algorithm does not work correctly for all ARG topologies (see Figure 8.5 for an example). As a workaround, we may consolidate only the closest segment of the recombinant sequence and replace the consolidated sequence in the recombinant and the parental sequence. However, it is more difficult to determine if the recombinant sequence can be removed from the alignment in this case, as we have to keep track which parts are already consolidated.
Even though the algorithm is a simple heuristic and potentially introduces many errors, it has several interesting characteristics compared to other approaches that reconstruct ARGs. Other approaches usually estimate local coalescent trees and infer recombination events as SPR operations between the local trees (see section 4.3.6). The proposed algorithm reconstructs the ARG bottom up instead, starting with the most recent coalescent event and ideally progressing towards older coalescent events. Such an approach is particularly attractive as we can probably estimate recent coalescent events more accurately than ancient coalescent events. But a bottom-up approach also comes at a cost: we cannot find the global optimum and may not detect weak and old recombination signals very well.

8.3 Summary

Recco constitutes a novel and interesting approach for analyzing recombination events in a sequence alignment. The underlying optimization problem was carefully chosen to reflect the common restriction of the recombination process in biology and does not model recombination events that introduce insertions or deletions. Furthermore, this restriction

also allows for a simple visualization of the set of α-optimal paths. But the strength of Recco's approach is mainly founded in the analysis of the parametric cost curve and in the Savings value as an intuitive measure of the recombination signal. The correlation of the Savings value with the strength of the recombination signal can even be directly derived from the ARG and provides a theoretical motivation for the approach of Recco. The recombination detection performance of Recco on simulated and real alignments supports that Recco also works well in practice.

Nevertheless, Recco has several inherent limitations. Recco only analyzes the most recent coalescent events in the ARG and therefore cannot easily detect ancient recombination events or recombination events that spawn more than one sequence in the alignment. Recco also suffers from the general deficiencies of a parsimony-based approach in phylogenetics and does not accurately model mutation events. However, the most relevant limitation in practice is the difficulty of scoring gaps appropriately.

It is my earnest hope that Recco provides a valuable and different view of the problem of recombination analysis and stimulates the investigation of new approaches for analyzing recombinant sequences.

9 References

Ahlenstiel, G., K. Roomp, et al. (2007). "Selective pressures of HLA genotypes and antiviral therapy on human immunodeficiency virus type 1 sequence mutation at a population level." Clinical and Vaccine Immunology **14**(10): 1266-1273.

Althaus, C. L. and S. Bonhoeffer (2005). "Stochastic Interplay between Mutation and Recombination during the Acquisition of Drug Resistance Mutations in Human Immunodeficiency Virus Type 1." Journal of Virology **79**(21): 13572-13578.

Altschul, S. F. (1991). "Amino acid substitution matrices from an information theoretic perspective." Journal of Molecular Biology **219**(3): 555-565.

Altschul, S. F., W. Gish, et al. (1990). "Basic local alignment search tool." Journal of Molecular Biology **215**(3): 403-410.

An, W. and A. Telesnitsky (2002). "HIV-1 genetic recombination: experimental approaches and observations." AIDS Reviews **4**(4): 195-212.

Azevedo, R. B. R., R. Lohaus, et al. (2006). "Sexual reproduction selects for robustness and negative epistasis in artificial gene networks." Nature **440**(7080): 87-90.

Bafna, V. and V. Bansal (2004). "The number of recombination events in a sample history: conflict graph and lower bounds." IEEE/ACM Transactions on Computational Biology and Bioinformatics **1**(2): 78-90.

Baird, H., Y. Gao, et al. (2006). "Influence of sequence identity and unique breakpoints on the frequency of intersubtype HIV-1 recombination." Retrovirology **3**(1): 91.

Bamshad, M. and S. P. Wooding (2003). "Signatures of natural selection in the human genome." Nature Reviews Genetics **4**(2): 99-111.

Bannert, N. and R. Kurth (2004). "Retroelements and the human genome: new perspectives on an old relation." Proceedings of The National Academy Of Sciences Of The United States Of America **101 Suppl 2**: 14572-14579.

Belshaw, R., V. Pereira, et al. (2004). "Long-term reinfection of the human genome by endogenous retroviruses." Proceedings of The National Academy Of Sciences Of The United States Of America **101**(14): 4894-4899.

Benjamini, Y. and Y. Hochberg (1995). "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing." Journal of the Royal Statistical Society Series B **57**(1): 289-300.

Bhattacharya, T., M. Daniels, et al. (2007). "Founder effects in the assessment of HIV polymorphisms and HLA allele associations." Science **315**(5818): 1583-1586.

Bonhoeffer, S., C. Chappey, et al. (2004). "Evidence for positive epistasis in HIV-1." Science **306**(5701): 1547-1550.

Bromham, L. and D. Penny (2003). "The modern molecular clock." Nature Reviews Genetics **4**(3): 216-224.

Brown, C. J., E. C. Garner, et al. (2001). "The power to detect recombination using the coalescent." Molecular Biology and Evolution **18**(7): 1421-1424.

Bruen, T. C., H. Philippe, et al. (2006). "A simple and robust statistical test for detecting the presence of recombination." Genetics **172**(4): 2665-2681.

Cartwright, R. A. (2005). "DNA assembly with gaps (Dawg): simulating sequence evolution." Bioinformatics **21 Suppl 3**: iii31-iii38.

Chamary, J. V., J. L. Parmley, et al. (2006). "Hearing silence: non-neutral evolution at synonymous sites in mammals." Nature Reviews Genetics **7**(2): 98-108.

Chenna, R., H. Sugawara, et al. (2003). "Multiple sequence alignment with the Clustal series of programs." Nucleic Acids Research **31**(13): 3497-3500.

Cheynier, R., L. Kils-Hütten, et al. (2001). "Insertion/deletion frequencies match those of point mutations in the hypervariable regions of the simian immunodeficiency virus surface envelope gene." Journal of General Virology **82**(Pt 7): 1613-1619.
Consortium, I. H., K. A. Frazer, et al. (2007). "A second generation human haplotype map of over 3.1 million SNPs." Nature **449**(7164): 851--861.
Cooper, T. F. (2007). "Recombination Speeds Adaptation by Reducing Competition between Beneficial Mutations in Populations of Escherichia coli." PLoS Biology **5**(9): e225.
D'Souza, V. and M. F. Summers (2005). "How retroviruses select their genomes." Nature Reviews Microbiology **3**(8): 643-655.
Domschke, W. and A. Drexl (1998). Einführung in Operations Research, Springer.
Drake, J. W., B. Charlesworth, et al. (1998). "Rates of spontaneous mutation." Genetics **148**(4): 1667-1686.
Edgar, R. C. and S. Batzoglou (2006). "Multiple sequence alignment." Current Opinion In Structural Biology **16**(3): 368-373.
Eisner, M. J. and D. G. Severance (1976). "Mathematical Techniques for Efficient Record Segmentation in Large Shared Databases." Journal of the ACM **23**(4): 619--635.
Etherington, G. J., J. Dicks, et al. (2005). "Recombination Analysis Tool (RAT): a program for the high-throughput detection of recombination." Bioinformatics **21**(3): 278-281.
Ewens, W. J. and G. R. Grant (2001). Statistical Methods in Bioinformatics, Springer.
Fang, F., J. Ding, et al. (2006). "cBrother: relaxing parental tree assumptions for Bayesian recombination detection." Bioinformatics.
Fearnhead, P. and P. Donnelly (2001). "Estimating recombination rates from population genetic data." Genetics **159**(3): 1299-1318.
Felsenstein, J. (1978). "The number of evolutionary trees." Systematic Zoology **27**: 27-33.
Felsenstein, J. (2004). Inferring Phylogenies, Sinauer Associates.
Flockerzi, A., J. Maydt, et al. (2007). "Expression pattern analysis of transcribed HERV sequences is complicated by ex vivo recombination." Retrovirology **4**: 39.
Flockerzi, A., A. Ruggieri, et al. (2008). "Expression patterns of transcribed human endogenous retrovirus HERV-K(HML-2) loci in human tissues and the need for a HERV Transcriptome Project." BMC Genomics **9**: 354.
Forslund, K., D. H. Huson, et al. (2004). "VisRD-visual recombination detection." Bioinformatics **20**(18): 3654-3655.
Foulds, L. R. and R. L. Graham (1982). "The Steiner problem in phylogeny is NP-complete." Advances in Applied Mathematics **3**: 43-49.
Fraser, C. (2005). "HIV recombination: what is the impact on antiretroviral therapy?" Journal of the Royal Society Interface **2**(5): 489-503.
Freed, E. O. (2001). "HIV-1 replication." Somatic Cell and Molecular Genetics **26**(1-6): 13-33.
Freed, E. O. (2004). "HIV-1 and the host cell: an intimate association." Trends In Microbiology **12**(4): 170-177.
Gale, C. V., R. Myers, et al. (2004). "Development of a novel human immunodeficiency virus type 1 subtyping tool, Subtype Analyzer (STAR): analysis of subtype distribution in London." AIDS Research and Human Retroviruses **20**(5): 457-464.
Galetto, R. and M. Negroni (2005). "Mechanistic features of recombination in HIV." AIDS Reviews **7**(2): 92-102.
Gifford, R. and M. Tristem (2003). "The evolution, distribution and diversity of endogenous retroviruses." Virus Genes **26**(3): 291-315.

Gogarten, J. P. and J. P. Townsend (2005). "Horizontal gene transfer, genome innovation and evolution." Nature Reviews Microbiology **3**(9): 679-687.
Gould, E. A. and T. Solomon (2008). "Pathogenic flaviviruses." Lancet **371**(9611): 500-509.
Grassly, N. C., P. H. Harvey, et al. (1999). "Population dynamics of HIV-1 inferred from gene sequences." Genetics **151**(2): 427-438.
Grassly, N. C. and E. C. Holmes (1997). "A likelihood method for the detection of selection and recombination using nucleotide sequences." Molecular Biology and Evolution **14**(3): 239-247.
Griffiths, R. C. and P. Marjoram (1996). "Ancestral inference from samples of DNA sequences with recombination." Journal of Computational Biology **3**(4): 479-502.
Griffiths, R. C. and P. Marjoram (1997). An ancestral recombination graph. Progress in Population Genetics and Human Evolution. P. Donnelly and S. Tavaré, Springer-Verlag, Berlin. **87:** 257-270.
Hastie, T., R. Tibshirani, et al. (2001). The Elements of Statistical Learning, Springer.
Heeney, J. L., A. G. Dalgleish, et al. (2006). "Origins of HIV and the evolution of resistance to AIDS." Science **313**(5786): 462-466.
Hein, J. (1990). "Reconstructing evolution of sequences subject to recombination using parsimony." Mathematical Biosciences **98**(2): 185-200.
Hein, J. (1993). "A heuristic method to reconstruct the history of sequences subject to recombination." Journal of Molecular Evolution **36**(4): 396-405.
Hein, J., M. H. Schierup, et al. (2005). Gene Genealogies, Variation and Evolution, Oxford University Press.
Heitman, J. (2006). "Sexual reproduction and the evolution of microbial pathogens." Current Biology **16**(17): R711-R725.
Henikoff, S. and J. G. Henikoff (1992). "Amino acid substitution matrices from protein blocks." Proceedings of The National Academy Of Sciences Of The United States Of America **89**(22): 10915-10919.
Heyer, W. D., K. T. Ehmsen, et al. (2003). "Holliday junctions in the eukaryotic nucleus: resolution in sight?" Trends In Biochemical Sciences **28**(10): 548-557.
Holland, B. R., K. T. Huber, et al. (2002). "Delta plots: a tool for analyzing phylogenetic distance data." Molecular Biology and Evolution **19**(12): 2051-2059.
Hudson, R. R. (1983). "Properties of a neutral allele model with intragenic recombination." Theoretical Population Biology **23**(2): 183-201.
Hudson, R. R. (2001). "Two-locus sampling distributions and their application." Genetics **159**(4): 1805-1817.
Hudson, R. R. (2002). "Generating samples under a Wright-Fisher neutral model of genetic variation." Bioinformatics **18**(2): 337-338.
Husmeier, D. (2005). "Discriminating between rate heterogeneity and interspecific recombination in DNA sequence alignments with phylogenetic factorial hidden Markov models." Bioinformatics **21 Suppl 2**: ii166-ii172.
Husmeier, D., R. Dybowski, et al., Eds. (2005). Probabilistic Modeling in Bioinformatics and Medical Informatics, Springer.
Husmeier, D. and G. McGuire (2002). "Detecting recombination with MCMC." Bioinformatics **18 Suppl 1**: S345-S353.
Husmeier, D. and G. McGuire (2003). "Detecting recombination in 4-taxa DNA sequence alignments with Bayesian hidden Markov models and Markov chain Monte Carlo." Molecular Biology and Evolution **20**(3): 315-337.
Husmeier, D. and F. Wright (2001). "Detection of recombination in DNA multiple alignments with hidden Markov models." Journal of Computational Biology **8**(4): 401-427.

Husmeier, D. and F. Wright (2001). "Probabilistic divergence measures for detecting interspecies recombination." Bioinformatics **17 Suppl 1**: S123-S131.
Husmeier, D., F. Wright, et al. (2005). "Detecting interspecific recombination with a pruned probabilistic divergence measure." Bioinformatics **21**(9): 1797-1806.
Huson, D. H. and D. Bryant (2006). "Application of Phylogenetic Networks in Evolutionary Studies." Molecular Biology and Evolution **23**(2): 254-267.
Jakobsen, I. B. and S. Easteal (1996). "A program for calculating and displaying compatibility matrices as an aid in determining reticulate evolution in molecular sequences." Computer Applications In The Biosciences **12**(4): 291-295.
Jakobsen, I. B., S. R. Wilson, et al. (1997). "The partition matrix: exploring variable phylogenetic signals along nucleotide sequence alignments." Molecular Biology and Evolution **14**(5): 474-484.
Jeffreys, A. J., L. Kauppi, et al. (2001). "Intensely punctate meiotic recombination in the class II region of the major histocompatibility complex." Nature Genetics **29**(2): 217-22.
Jetzt, A. E., H. Yu, et al. (2000). "High rate of recombination throughout the human immunodeficiency virus type 1 genome." Journal of Virology **74**(3): 1234-1240.
Johnson, W. E. and J. M. Coffin (1999). "Constructing primate phylogenies from ancient retrovirus sequences." Proceedings of The National Academy Of Sciences Of The United States Of America **96**(18): 10254-10260.
Jukes, T. H. and C. R. Cantor (1969). Evolution of protein molecules. Mammalian Protein Metabolism. H. N. Munro**:** 21-132.
Jung, A., R. Maier, et al. (2002). "Multiply infected spleen cells in HIV patients." Nature **418**(6894): 144.
Kececioglu, J. D. and D. Gusfield (1998). "Reconstructing a History of Recombinations From a Set of Sequences." Discrete Applied Mathematics **88**(1-3): 239-260.
Kedzierska, A. and D. Husmeier (2006). "A heuristic Bayesian method for segmenting DNA sequence alignments and detecting evidence for recombination and gene conversion." Statistical Applications in Genetics and Molecular Biology **5**: Article 27.
Keele, B. F., F. V. Heuverswyn, et al. (2006). "Chimpanzee reservoirs of pandemic and nonpandemic HIV-1." Science **313**(5786): 523-526.
Kijak, G. H. and F. E. McCutchan (2005). "HIV Diversity, Molecular Epidemiology, and the Role of Recombination." Current Infectious Disease Reports **7**(6): 480-488.
Kimura, M. (1968). "Evolutionary Rate at the Molecular Level." Nature **217**: 624-626.
Kimura, M. (1980). "A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences." Journal of Molecular Evolution **16**(2): 111-120.
Kimura, M. (1991). "The neutral theory of molecular evolution: a review of recent evidence." The Japanese Journal of Genetics **66**(4): 367-386.
King, J. L. and T. H. Jukes (1969). "Non-Darwinian evolution." Science **164**(881): 788-798.
Kingman, J. F. (1982). "On the genealogy of large populations." Journal of Applied Probability: 27-43.
Kong, A., D. F. Gudbjartsson, et al. (2002). "A high-resolution recombination map of the human genome." Nature Genetics **31**(3): 241-247.
Kuhner, M. K., J. Yamato, et al. (2000). "Maximum likelihood estimation of recombination rates from population data." Genetics **156**(3): 1393-1401.
Lajoie, M. and N. El-Mabrouk (2005). "Recovering haplotype structure through recombination and gene conversion." Bioinformatics **21 Suppl 2**: ii173-ii179.

Levy, D. N., G. M. Aldrovandi, et al. (2004). "Dynamics of HIV-1 recombination in its natural target cells." Proceedings of The National Academy Of Sciences Of The United States Of America **101**(12): 4204-4209.
Li, W. H. (1997). Molecular Evolution, Sinauer Associates.
Lodish, H., A. Berk, et al., Eds. (2001). Molecular Cell Biology, W. H. Freeman and Company.
Lole, K. S., R. C. Bollinger, et al. (1999). "Full-length human immunodeficiency virus type 1 genomes from subtype C-infected seroconverters in India, with evidence of intersubtype recombination." Journal of Virology **73**(1): 152-160.
Löwer, R., J. Löwer, et al. (1996). "The viruses in all of us: characteristics and biological significance of human endogenous retrovirus sequences." Proceedings of The National Academy Of Sciences Of The United States Of America **93**(11): 5177-5184.
Lunter, G. A., I. Miklós, et al. (2003). "An efficient algorithm for statistical multiple alignment on arbitrary phylogenetic trees." Journal of Computational Biology **10**(6): 869-889.
Luo, G. X. and J. Taylor (1990). "Template switching by reverse transcriptase during DNA synthesis." Journal of Virology **64**(9): 4321-4328.
Magiorkinis, G., D. Paraskevis, et al. (2005). "Re-analysis of 34 full-length HIV-1 intersubtype recombinant sequences." Infection, Genetics and Evolution **5**(3): 225-229.
Mallet, J. (2007). "Hybrid speciation." Nature **446**(7133): 279-283.
Mansky, L. M. (1998). "Retrovirus mutation rates and their role in genetic variation." Journal of General Virology **79 (Pt 6)**: 1337-1345.
Mansky, L. M. and H. M. Temin (1995). "Lower in vivo mutation rate of human immunodeficiency virus type 1 than that predicted from the fidelity of purified reverse transcriptase." Journal of Virology **69**(8): 5087-5094.
Martin, D. and E. Rybicki (2000). "RDP: detection of recombination amongst aligned sequences." Bioinformatics **16**(6): 562-563.
Martin, D. P., C. Williamson, et al. (2005). "RDP2: recombination detection and analysis from sequence alignments." Bioinformatics **21**(2): 260-262.
Maydt, J. and T. Lengauer (2006). "Recco: recombination analysis using cost optimization." Bioinformatics: 1064-1071.
McBreen, K. and P. J. Lockhart (2006). "Reconstructing reticulate evolutionary histories of plants." Trends in Plant Science **11**(8): 398-404.
McGuire, G. and F. Wright (1998). "TOPAL: recombination detection in DNA and protein sequences." Bioinformatics **14**(2): 219-220.
McGuire, G. and F. Wright (2000). "TOPAL 2.0: improved detection of mosaic sequences within multiple alignments." Bioinformatics **16**(2): 130-134.
McVean, G. A. T., S. R. Myers, et al. (2004). "The fine-scale structure of recombination rate variation in the human genome." Science **304**(5670): 581-584.
Mevissen, H. T. and M. Vingron (1996). "Quantifying the local reliability of a sequence alignment." Protein Engineering **9**(2): 127-132.
Meyer, S. and A. von Haeseler (2003). "Identifying site-specific substitution rates." Molecular Biology and Evolution **20**(2): 182-189.
Meyerhans, A., J. P. Vartanian, et al. (1990). "DNA recombination during PCR." Nucleic Acids Research **18**(7): 1687-1691.
Milne, I., F. Wright, et al. (2004). "TOPALi: software for automatic identification of recombinant sequences within DNA multiple alignments." Bioinformatics **20**(11): 1806-1807.
Minichiello, M. J. and R. Durbin (2006). "Mapping trait Loci by use of inferred ancestral recombination graphs." American Journal of Human Genetics **79**(5): 910-922.

Minin, V. N., K. S. Dorman, et al. (2005). "Dual multiple change-point model leads to more accurate recombination detection." Bioinformatics: 3034-3042.
Moutouh, L., J. Corbeil, et al. (1996). "Recombination leads to the rapid emergence of HIV-1 dually resistant mutants under selective drug pressure." Proceedings of The National Academy Of Sciences Of The United States Of America **93**(12): 6106-6111.
Mu, J., P. Awadalla, et al. (2005). "Recombination hotspots and population structure in Plasmodium falciparum." PLoS Biology **3**(10): e335.
Myers, R. E., C. V. Gale, et al. (2005). "A statistical model for HIV-1 sequence classification using the subtype analyser (STAR)." Bioinformatics **21**(17): 3535-3540.
Myers, S. R. and R. C. Griffiths (2003). "Bounds on the minimum number of recombination events in a sample history." Genetics **163**(1): 375-394.
Nakhleh, L., T. Warnow, et al. (2005). "Reconstructing reticulate evolution in species-theory and practice." Journal of Computational Biology **12**(6): 796-811.
Negroni, M. and H. Buc (2001). "Mechanisms of retroviral recombination." Annual Review of Genetics **35**: 275-302.
Negroni, M. and H. Buc (2001). "Retroviral recombination: what drives the switch?" Nature Reviews Molecular Cell Biology **2**(2): 151-155.
Nisole, S. and A. Saïb (2004). "Early steps of retrovirus replicative cycle." Retrovirology **1**: 9.
Ogden, T. H. and M. S. Rosenberg (2006). "How should gaps be treated in parsimony? A comparison of approaches using simulation." Molecular Phylogenetics and Evolution.
Ogden, T. H. and M. S. Rosenberg (2006). "Multiple sequence alignment accuracy and phylogenetic inference." Systematic Biology **55**(2): 314-328.
Ohta, T. (2002). "Near-neutrality in evolution of genes and gene regulation." Proceedings of The National Academy Of Sciences Of The United States Of America **99**(25): 16134-16137.
Otto, S. P. and T. Lenormand (2002). "Resolving the paradox of sex and recombination." Nature Reviews Genetics **3**(4): 252-261.
Paraskevis, D., K. Deforche, et al. (2005). "SlidingBayes: exploring recombination using a sliding window approach based on Bayesian phylogenetic inference." Bioinformatics **21**(7): 1274-1275.
Pesole, G. and C. Saccone (2001). "A novel method for estimating substitution rate variation among sites in a large dataset of homologous DNA sequences." Genetics **157**(2): 859-865.
Pond, S. L. K., D. Posada, et al. (2006). "Automated Phylogenetic Detection of Recombination Using a Genetic Algorithm." Molecular Biology and Evolution **23**(10): 1891-1901.
Pond, S. L. K., D. Posada, et al. (2006). "GARD: a genetic algorithm for recombination detection." Bioinformatics **22**(24): 3096-3098.
Posada, D. (2002). "Evaluation of methods for detecting recombination from DNA sequences: empirical data." Molecular Biology and Evolution **19**(5): 708-717.
Posada, D. and K. A. Crandall (2001). "Evaluation of methods for detecting recombination from DNA sequences: computer simulations." Proceedings of The National Academy Of Sciences Of The United States Of America **98**(24): 13757-13762.
Rambaut, A. and N. C. Grassly (1997). "Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees." Computer Applications In The Biosciences **13**(3): 235-238.
Rambaut, A., D. Posada, et al. (2004). "The causes and consequences of HIV evolution." Nature Reviews Genetics **5**(1): 52-61.

Robertson, D. L., J. P. Anderson, et al. (2000). "HIV-1 nomenclature proposal." Science **288**(5463): 55-56.
Rubbert, A., G. Behrens, et al. (2007). HIV Medicine. C. Hoffmann, J. K. Rockstroh and B. S. Kamps, Flying Publisher**:** 59-86.
Salminen, M. O., J. K. Carr, et al. (1995). "Identification of breakpoints in intergenotypic recombinants of HIV type 1 by bootscanning." AIDS Research and Human Retroviruses **11**(11): 1423-1425.
Sawyer, S. (1989). "Statistical tests for detecting gene conversion." Molecular Biology and Evolution **6**(5): 526-538.
Schierup, M. H. and J. Hein (2000). "Consequences of recombination on traditional phylogenetic analysis." Genetics **156**(2): 879-891.
Schultz, A.-K., M. Zhang, et al. (2006). "A jumping profile Hidden Markov Model and applications to recombination sites in HIV and HCV genomes." BMC Bioinformatics **7**(1): 265.
Schwikowski, B. (1998). A New Algorithmic Approach to the Construction of Multiple Alignments and Evolutionary Trees, GMD - Forschungszentrum Informationstechnik GmbH.
Siepel, A. C., A. L. Halpern, et al. (1995). "A computer program designed to screen rapidly for HIV type 1 intersubtype recombinant sequences." AIDS Research and Human Retroviruses **11**(11): 1413-1416.
Simmons, M. P. and H. Ochoterena (2000). "Gaps as characters in sequence-based phylogenetic analyses." Systematic Biology **49**(2): 369-381.
Simonsen, K. L., G. A. Churchill, et al. (1995). "Properties of statistical tests of neutrality for DNA polymorphism data." Genetics **141**(1): 413-429.
Smith, J. M. (1992). "Analyzing the mosaic structure of genes." Journal of Molecular Evolution **34**(2): 126-129.
Smith, N. G. C. and P. Fearnhead (2005). "A Comparison of Three Estimators of the Population-scaled Recombination Rate: Accuracy and Robustness." Genetics **171**: 2051-2062.
Sniegowski, P. D., P. J. Gerrish, et al. (2000). "The evolution of mutation rates: separating causes from consequences." Bioessays **22**(12): 1057-1066.
Song, Y. S. (2006). "Properties of Subtree-Prune-and-Regraft Operations on Totally-Ordered Phylogenetic Trees." Annals of Combinatorics **10**: 129-146.
Song, Y. S. and J. Hein (2003). Parsimonious Reconstruction of Sequence Evolution and Haplotype Blocks. WABI.
Song, Y. S. and J. Hein (2005). "Constructing minimal ancestral recombination graphs." Journal of Computational Biology **12**(2): 147-169.
Song, Y. S., Y. Wu, et al. (2005). "Efficient computation of close lower and upper bounds on the minimum number of recombinations in biological sequence evolution." Bioinformatics **21 Suppl 1**: i413-i422.
Spang, R., M. Rehmsmeier, et al. (2000). "Sequence database search using jumping alignments." Proceedings of the Eighth International Conference on Intelligent systems for Molecular Biology **8**: 367-375.
Spang, R., M. Rehmsmeier, et al. (2002). "A novel approach to remote homology detection: jumping alignments." Journal of Computational Biology **9**(5): 747-760.
Spencer, M. (2003). "Exact significance levels for the maximum chi(2) method of detecting recombination." Bioinformatics **19**(11): 1368-1370.
Spratt, B. G., W. P. Hanage, et al. (2001). "The relative contributions of recombination and point mutation to the diversification of bacterial clones." Current Opinion in Microbiology **4**(5): 602-606.
Strimmer, K., K. Forslund, et al. (2003). "A novel exploratory method for visual recombination detection." Genome Biology **4**(5): R33.

Stumpf, M. P. H. and G. A. T. McVean (2003). "Estimating recombination rates from population-genetic data." Nature Reviews Genetics **4**(12): 959-968.
Suchard, M. A., R. E. Weiss, et al. (2002). "Oh brother, where art thou? A Bayes factor test for recombination with uncertain heritage." Systematic Biology **51**(5): 715-728.
Suchard, M. A., R. E. Weiss, et al. (2003). "Inferring Spatial Phylogenetic Variation Along Nucleotide Sequences: A Multiple Changepoint Model." Journal of the American Statistical Association **98**: 427-437.
Suryavanshi, G. W. and N. M. Dixit (2007). "Emergence of recombinant forms of HIV: dynamics and scaling." PLoS Computational Biology **3**(10): 2003-2018.
Takahata, N. (1996). "Neutral theory of molecular evolution." Current Opinion In Genetics and Development **6**(6): 767-772.
Tavaré, S. (2004). Lectures on Probability Theory and Statistics, Springer**:** 1-188.
Thomas, C. M. and K. M. Nielsen (2005). "Mechanisms of, and barriers to, horizontal gene transfer between bacteria." Nature Reviews Microbiology **3**(9): 711-721.
Thomson, M. M. and R. Nájera (2005). "Molecular epidemiology of HIV-1 variants in the global AIDS pandemic: an update." AIDS Reviews **7**(4): 210-224.
Turner, G., M. Barbulescu, et al. (2001). "Insertional polymorphisms of full-length endogenous retroviruses in humans." Current Biology **11**(19): 1531-1535.
Wang, L., B. Ma, et al. (2000). "Fixed topology alignment with recombination." Discrete Applied Mathematics **104**: 281-300.
Wang, L., K. Zhang, et al. (2001). "Perfect phylogenetic networks with recombination." Journal of Computational Biology **8**(1): 69-78.
Weiller, G. F. (1998). "Phylogenetic profiles: a graphical method for detecting genetic recombinations in homologous sequences." Molecular Biology and Evolution **15**(3): 326-335.
Wiuf, C., T. Christensen, et al. (2001). "A simulation study of the reliability of recombination detection methods." Molecular Biology and Evolution **18**(10): 1929-1939.
Wiuf, C. and J. Hein (1999). "Recombination as a point process along sequences." Theoretical Population Biology **55**(3): 248-259.
Wu, Y. (2007). Association Mapping of Complex Diseases with Ancestral Recombination Graphs: Models and Efficient Algorithms. Research in Computational Molecular Biology, Springer. **4453:** 488-502.
Yang, Z. (1996). "Among-site rate variation and its impact on phylogenetic analyses." Trends in Ecology & Evolution **11**: 367-372.
Young, N. D. and J. Healy (2003). "GapCoder automates the use of indel characters in phylogenetic analysis." BMC Bioinformatics **4**: 6.
Zhang, M., A.-K. Schultz, et al. (2006). "jpHMM at GOBICS: a web server to detect genomic recombinations in HIV-1." Nucleic Acids Research **34**(Web Server issue): W463-W465.
Zimmer, R. and T. Lengauer (1997). Fast and numerically stable parametric alignment of biosequences. RECOMB '97: Proceedings of the first annual international conference on Computational molecular biology, New York, NY, USA, ACM Press.