Department of Computer Science
Faculty of Mathematics, Physics, and Informatics
Comenius University, Bratislava, Slovakia

Algorithms for Genome Rearrangements

Jakub Kováč

A thesis presented to the Comenius University
in fulfillment of the thesis requirement for the degree of
Doctor of Philosophy in Computer Science

Supervisor: Mgr. Tomáš Vinař, PhD.

Bratislava, 2013

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Mgr. Jakub Kováč
Study programme:	Computer Science (Single degree study, Ph.D. III. deg., full time form)
Field of Study:	9.2.1. Computer Science, Informatics
Type of Thesis:	Dissertation thesis
Language of Thesis:	English

Title: Algorithms for Genome Rearrangements

Aim: The goal of this work is to find solutions for small phylogeny, median, and halving problems on several mathematical models of genome rearrangements. The work should study these problems both from the theoretical perspective (complexity and algorithms) and from the practical point of view (practical solutions for heterogeneous data sets).

Tutor:	Mgr. Tomáš Vinař, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	doc. RNDr. Daniel Olejár, PhD.
Assigned:	19.10.2010

Approved: 19.10.2010 prof. RNDr. Branislav Rovan, PhD.
Guarantor of Study Programme

.....
Student

.....
Tutor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Mgr. Jakub Kováč
Študijný program: informatika (Jednoodborové štúdium, doktorandské III. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: dizertačná
Jazyk záverečnej práce: anglický

Názov: Algoritmy pre problémy preusporiadania genómov

Cieľ: Cieľom práce je riešenie problémov rekonštrukcie predkov, mediánu a polenia na niekoľkých matematických modeloch preusporiadania genómov. Práca bude študovať problémy z teoretickej stránky (zložitosť a algoritmy), ako aj z pohľadu praktickej aplikácie (praktické riešenia pre heterogénne dáta).

Školiteľ: Mgr. Tomáš Vinař, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 19.10.2010

Dátum schválenia: 19.10.2010 prof. RNDr. Branislav Rovan, PhD.
garant študijného programu

.....
študent

.....
školiteľ

Acknowledgements

I am very grateful to Tomáš Vinař and Broňa Brejová for their guidance, encouragement, and introducing me to the world of biology. I would also like to thank my coauthors Marília Braga and Jens Stoye.

It has been pleasure to work with you.

Contents

Abstract (English)

In this thesis, we study several algorithmic problems from the field of genome rearrangements. During evolution, genomes undergo large-scale mutations. A segment of DNA can get reversed, moved to another position, or even another chromosome.

If we compare genomes of related extant species, we can find long conserved regions of DNA (such as genes) which are very similar sequentially, however their order is different. This motivates the following biological problems which also pose intriguing challenges for computer science:

- How related are the two given organisms?
- How did their ancestor look like?
- More generally: If we know gene orders of multiple species and their phylogenetic tree, how did the ancestral genomes look like?
- Given just the gene orders of multiple species, can we reconstruct their phylogenetic tree?

We formulate these questions as optimization problems: assuming a genome model with a fixed set of allowed rearrangement operations, we can define distance between two genomes as the minimum number of rearrangements necessary to transform one genome into the other. The problems of reconstructing the evolutionary history and the phylogenetic tree of given species is also formulated using the parsimony criterion: we search a phylogenetic tree and ancestral genomes which minimize the total number of rearrangement mutations in the evolutionary history.

In this thesis, we are interested in both theoretical and practical problems in genome rearrangements. We propose a new approach to ancestral genome reconstruction and we implement one of the first practical tools applicable to analysis of real datasets spanning a complex phylogeny and accommodating a variety of genome architectures. We demonstrate the accuracy of our program on the well-studied dataset of *Campanulaceae* chloroplast genomes, and apply it to the reconstruction of rearrangement histories of newly sequenced mitochondrial genomes of pathogenic yeasts from *Hemiascomycetes* clade.

We revisit the restricted DCJ model by Yancopoulos et al. We propose an $O(n \log n)$ time algorithm for sorting in this model, thus improving on the existing quadratic algorithm, and develop a new linear time algorithm for genome halving.

Our main results concern several open problems in the breakpoint model. We give an $O(n\sqrt{n})$ algorithm for the median problem improving on the existing cubic algorithm. Furthermore, we show that the problem is equivalent to finding maximum matching. Thus, any improvement to our solution would imply a better algorithm for the maximum matching, which has been an open problem for more than 30 years. We also prove that the more general small phylogeny problem is NP-hard. Surprisingly, we show that it is NP-hard (even APX-hard) already for four species. In other words, while finding an ancestor of three species is easy, finding two ancestors of four species is already hard. We thereby solve two open problems from the monograph by Fertin et al.: *Combinatorics of genome rearrangements.*

Abstrakt (Slovensky)

V dizertačnej práci sa zaoberáme preusporiadaniami génov rôznych organizmov. Počas evolúcie sa genómy organizmov menia a vyvíjajú (mutujú). Okrem drobných zmien, pri ktorých sa mení len jedna alebo zopár susedných báz (nukleotidov), sa počas evolúcie z času na čas stane, že sa nejaký dlhší úsek DNA presunie na iné miesto, na opačné vlákno, či iný chromozóm. Ak sa teda dnes pozrieme na genómy príbuzných druhov, vieme v nich nájsť veľmi podobné úseky, ktoré sú však v rôznych druhoch na rôznych miestach v genóme. To motivuje nasledujúce biologicky zaujímavé otázky, ktoré sú tiež výzvou pre teoretickú informatiku:

- Nakoľko sú si dva organizmy príbuzné?
- Ako asi vyzeral genóm ich spoločného predka?
- Alebo všeobecnejšie: Ak poznáme genómy viacerých druhov a ich fylogenetický strom, ako asi vyzerala ich evolučná história?
- Ak poznáme iba genómy viacerých druhov, ako vyzerá ich fylogenetický strom?

Existuje viacero matematických modelov, ktoré tieto javy podchycujú. Jednotlivé modely sa rôznia podľa toho, či uvažujeme druhy s jedným alebo možno viacerými chromozómami, či uvažujeme druhy s lineárnymi alebo cirkulárnymi chromozómami (alebo oboje) a tiež, aké typy mutácií uvažujeme: inverzie, transpozície, translokácie, atď. Vzdialenosť medzi dvoma genómami môžeme definovať ako minimálny počet operácií, ktoré preusporiadajú jeden genóm na druhý. Úlohy o rekonštrukcii evolučnej histórie potom formulujeme ako hľadanie „najúspornejšieho“ riešenia – pozorované poradia génov sa snažíme vysvetliť pomocou čo najmenšieho počtu mutácií.

V rámci dizertačnej práce sa venujeme viacerým teoretickým, ale aj praktickejším otázkam z oblasti preusporiadania genómov. Navrhli sme nový prístup k rekonštrukcii usporiadaní génov a implementovali sme jeden z prvých praktických nástrojov na rekonštrukciu evolučnej histórie druhov s rôznymi chromozómovými architektúrami. V rámci projektu, na ktorom sme spolupracovali s odborníkmi z Prírodovedeckej fakulty UK, sme náš program použili na rekonštrukciu usporiadaní génov kvasinkových mitochondriálnych genómov.

Navrhli sme nový, biologicky hodnovernejší variant modelu DCJ. V tomto modeli sme navrhli $O(n \log n)$ algoritmus pre tzv. problém triedenia, čím sme zlepšili dovtedy známy kvadratický algoritms, a vyriešili sme problém pólenia genómov.

Vyriešili sme tiež viacero otvorených teoretických problémov v breakpoint modeli: Navrhli sme $O(n\sqrt{n})$ algoritmus pre problém mediánu, čím sme zlepšili dovtedy známy kubický algoritmus. Tiež sme dokázali, že zlepšenie nášho algoritmu by viedlo k lepšiemu algoritmu pre hľadanie maximálneho párovania, čo je vyše 30 rokov otvorený problém. Následne sme sa venovali problému rekonštrukcie ancestrálnych usporiadaní genómov. Dokázali sme, že už pre štyri genómy je problém NP-ťažký, dokonca APX-ťažký. Poznamenajme, že sme tým vyriešili dva otvorené problémy z monografie Fertin a spol.: *Combinatorics of genome rearrangements.*

Chapter 1

Introduction

1.1 Genome Rearrangements: A Computer Scientist's Perspective

Introductory example. One of the first things a future computer scientist learns is how to sort a sequence of numbers efficiently (Knuth, 1973).

Now imagine that we are reordering heavy items, so we use a machine that can only exchange two elements at a time. Since this machine is quite slow, we would like to save time and sort the items using the least number of exchanges possible. Also, we would like to know the total number of steps needed (in order to know whether we have enough time for a coffee break). Consider for example permutation π on Fig. 1.1. How many exchanges does it take to sort the permutation?

The answer is quite simple: Draw an edge from each element i to π_i as in Fig. 1.1. Since each element has one incoming and one outgoing edge, the graph consists of cycles only. In general, any permutation can be uniquely decomposed into a set of cycles. Notice the self-loops at 5 and 8 – these signify that 5 and 8 are at their proper positions and do not need to be moved. In the identity permutation, all cycles are self-loops, so we can think about sorting as breaking cycles.

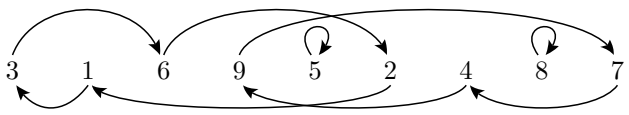
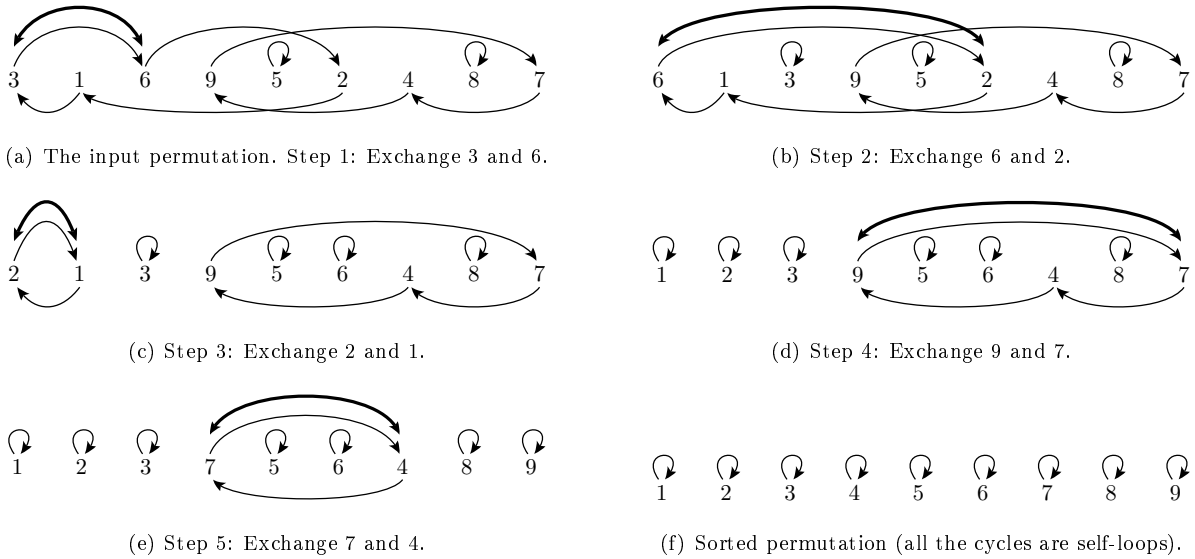


Figure 1.1: Cycles of permutation.

It is easy to see that by any single exchange, the number of cycles may increase by at most 1. Since our permutation π has only 4 cycles and the sorted permutation has 9 cycles, we need at least 5 exchanges.

On the other hand, in each step, we can move one element to its proper place and thus create a new self-loop, so 5 exchanges are sufficient (see Fig. 1.2 for a 5-step sorting scenario). In general, a permutation π on $\{1, 2, \ldots, n\}$, which can be decomposed into $c(\pi)$ cycles, can be sorted by $d(\pi) = n - c(\pi)$ exchanges.



(a) The input permutation. Step 1: Exchange 3 and 6.

(b) Step 2: Exchange 6 and 2.

(c) Step 3: Exchange 2 and 1.

(d) Step 4: Exchange 9 and 7.

(e) Step 5: Exchange 7 and 4.

(f) Sorted permutation (all the cycles are self-loops).

Figure 1.2: One way of sorting permutation $(3, 1, 6, 9, 5, 2, 4, 8, 7)$ by 5 exchanges.

We may ask a more general question: Given two permutations π and σ, what is the minimum number of exchanges $d(\pi, \sigma)$ needed to transform π into σ? We call $d(\pi, \sigma)$ the *distance* from π to σ and it can be easily shown that d is indeed a metric on the symmetric group S_n.

It should not be very surprising that computing the distance between two permutations is nothing else but sorting – up to renaming the elements. More precisely: $d(\pi, \sigma) = d(\sigma^{-1} \circ \pi, \sigma^{-1} \circ \sigma) = d(\sigma^{-1} \circ \pi, \iota) \equiv d(\sigma^{-1} \circ \pi)$. Furthermore, given two permutations π and σ, we can find their distance and a sequence of exchanges transforming π into σ in linear time.

Variants. Once we have solved the problem of sorting by exchanges, we may think about other variants of this problem. What if our machine did not exchange two elements? We may wonder, how would we sort our sequence, if the machine could, for example, take a block of elements of arbitrary length and insert it elsewhere into the sequence; or if the machine could exchange two blocks of arbitrary length.

By varying the available rearrangement operations, we obtain different metrics and different variants of the sorting problem. In each case, the problems are of the following form: Given two permutations π and σ,

- find the least number of operations transforming π into σ, and
- find a particular sequence of operations of the minimum length.

Sorting pancakes. One of the well known variants of the problem (at least in the computer science community) is the sorting by *prefix reversals*, or *pancake sorting* problem. Imagine a plate with a stack of pancakes. Due to a sloppy chef, all the pancakes come in different sizes and we would like to sort them

from the largest at the bottom to the smallest on the top. We can use a flipper to lift some pancakes from the top and flip them all at once. What is the minimum number of flips we need to sort the pancakes?

Probably the simplest way to sort the pancakes takes $2n-3$ steps: We proceed from bottom up; once the $(k-1)$ largest pancakes are at the bottom, we slip the flipper right beneath the k^{th} largest pancake. This flip moves the pancake to the top of the stack and with another flip, we can move it to its proper place.

Thus, we have an upper bound on the number of flips. However, this is rarely the shortest sorting sequence (see for example Fig. 1.3 showing a permutation of length 6 that can be sorted by 7 flips).

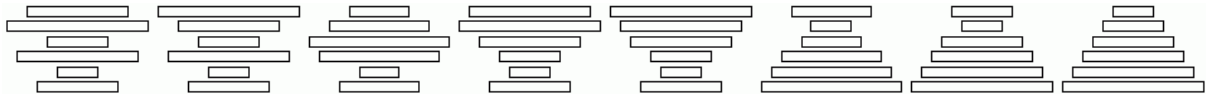


Figure 1.3: Sorting a stack of pancakes $(4, 6, 2, 5, 1, 3)$ by 7 flips. (This is one of the two permutations of length 6 that needs the highest number of flips.)

What can we say about a lower bound? Let us add element $n+1$ at the end of a permutation (set $\pi_{n+1} = n+1$). We call two successive elements π_i, π_{i+1} an *adjacency*, if π_i and π_{i+1} are two consecutive numbers ($|\pi_i - \pi_{i+1}| = 1$); otherwise call the pair a *breakpoint*. Identity permutation of length n has n adjacencies and no breakpoints. Furthermore, by one flip we can decrease the number of breakpoints by at most one, so the number of breakpoints in a permutation is a lower bound on the number of flips. For example, the permutation in Fig. 1.3 has 6 breakpoints (including the bottom one), so it requires at least 6 flips to sort.

These bounds are not very tight and certainly not the best bounds known. The number of flips for the worst-case permutation (diameter of the metric space) lies between $^{15}/_{14}n$ (Heydari and Sudborough, 1997) and $^{18}/_{11}n + O(1)$ (Chitturi et al., 2009). It has been recently shown that computing the exact distance is NP-hard (Bulteau et al., 2012a).

Sorting burnt pancakes. In the previous problem, both sides of each pancake were equivalent. In a related problem called *burnt pancakes*, the chef was even sloppier and all the pancakes are burnt on one side. In addition to sorting them from the largest to the smallest, we want all the pancakes to be placed burnt side down (so the customer does not notice). That is, in this case, the pancakes have *orientation* (burnt side up or burnt side down); flipping them reverses their order and flips their orientation.

We model burnt pancakes by *signed permutations*, where each element x has orientation $\overleftarrow{x}$ or $\overrightarrow{x}$. We write simply x if we do not refer to x's orientation and $-x$ for x with the opposite orientation (so $-\overleftarrow{x} = \overrightarrow{x}$ and $-\overrightarrow{x} = \overleftarrow{x}$).

There is not much known about sorting burnt pancakes. For instance, the complexity of computing the minimum number of flips for a given permutation and orientations of the pancakes is not known. For the diameter, we have a lower bound of $^{3}/_{2}n$ and an upper bound of $2n-2$ (Cohen and Blum, 1995).

Sorting by reversals. As a final example, let us present a problem, which is much better explored. Imagine that we have two flippers. When sorting the pancakes, we use the first one to lift a couple of pancakes from the top. Then we use the second one to actually flip some pancakes. Finally, we return the lifted pancakes back to the top of the pile in the original order and orientation. In this way, we can reverse any block of pancakes and the minimum number of these moves required to sort a permutation is called *reversal distance*.

The problem of computing reversal distance is solved for both signed and unsigned permutations and these results are one of the most profound results in the field of genome rearrangements. Even though the difference between unsigned and signed permutations may seem subtle, at least in case of reversal distance, the impact of this change on algorithmic solution is profound: while computing the unsigned reversal distance is NP-hard, not approximable within 1.0008, but 1.375-approximable (Caprara, 1997; Berman and Karpinski, 1999; Berman et al., 2002), the reversal distance of signed permutations can be found in linear time (Hannenhalli and Pevzner, 1999; Bader et al., 2001).

Genome rearrangements. Problems such as sorting pancakes and its variants belong to recreational mathematics and nobody ever expected that they would have any practical applications. A new impetus for the field, however, came from molecular biology and genetics where operations such as reversals or movements of entire blocks actually happen at the level of DNA. During the evolution, rearrangement mutations shuffle the order of genes in genomes of organisms and seeing the present-day state, we would like to infer something about the evolutionary history. Thus, we could say that today people spend more time sorting genes than pancakes.

In the next section, we review the basics of biology and all the interesting rearrangement operations and in the subsequent section, we state some problems inspired by comparative genomics that we will study in the rest of the thesis.

1.2 Genome Rearrangements: A Biologist's Perspective

Genome. All the hereditary information that is passed from parent to an offspring is stored in form of DNA and is present in every cell of a living organism (see Fig. 1.4). Regions of DNA called *genes* are "recipes" for construction of other macro-molecules in the cell: the RNAs and proteins, which are the workhorses of the cell.

DNA is made of two strands twisted in a double helix. Each strand is a long chain of repeating units, called *nucleotides* or *bases* (adenine, cytosine, guanine, or thymine), and we can represent the strand as a string over the alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. The two strands are complementary: each $\mathtt{A}$ on one strand is always paired with a $\mathtt{T}$ on the other strand and each $\mathtt{C}$ is always paired with a $\mathtt{G}$ on the other strand. Thus, each strand can serve as a template to create the other one, which is important during the replication of DNA. Moreover, a strand of DNA has a direction (this is due to the asymmetric nature of the chemical bonds linking nucleotides) and the two strands run in opposite directions.

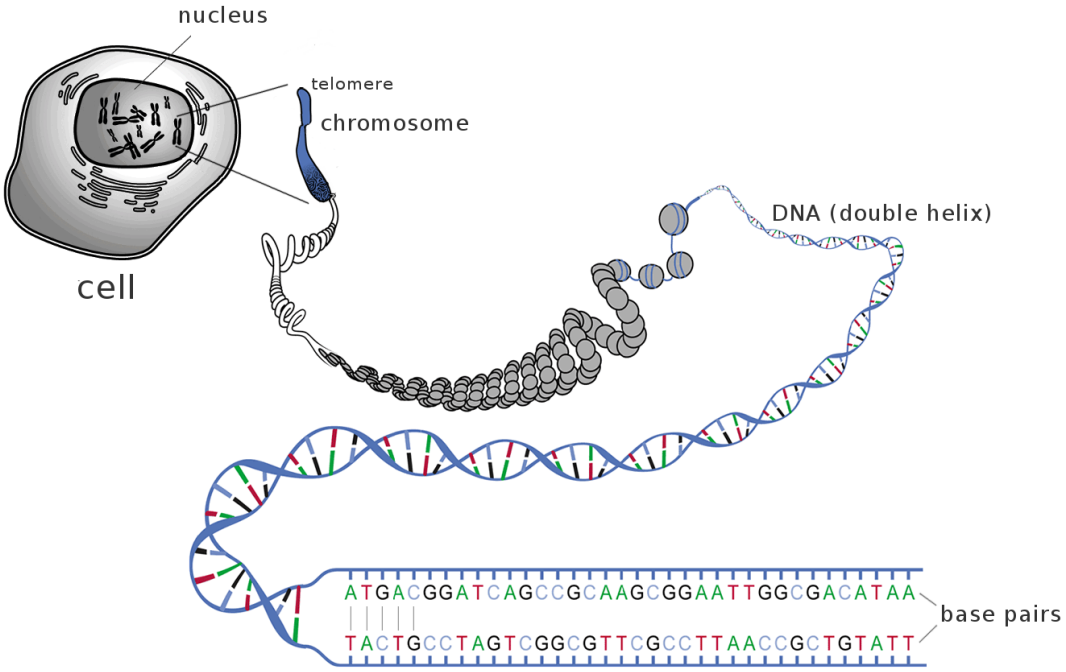
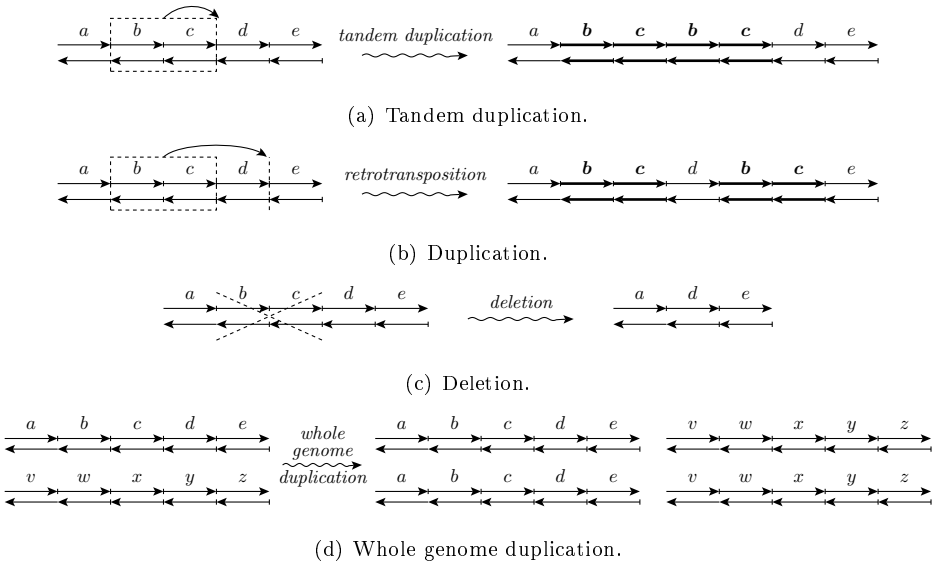


Figure 1.4: Nuclear genome consisting of multiple linear chromosomes. Each chromosome is a single long DNA molecule tightly packed together. DNA consists of two complementary strands running in opposite directions and twisted in a double helix. Each strand is a long chain of nucleotides: adenine, cytosine, guanine, and thymine (`A`, `C`, `G`, `T`); `A` is always paired with `T` and `C` is always paired with `G`. *Source:* modified from National Human Genome Research Institute.

Each DNA molecule is tightly packed together forming a structure called *chromosome* and the whole genome may consist of several chromosomes. While genomes of prokaryotes (bacteria and archaea) usually consist of just a single chromosome in their cytoplasm, cells of eukaryotic organisms (such as animals, plants, or fungi) typically contain multiple chromosomes in their nuclei. For example baker's yeasts have 32 chromosomes, mice have 40 chromosomes, and humans have 46 chromosomes. In eukaryotes, some DNA is also stored in the organelles such as mitochondria or chloroplasts.

Species may also differ in chromosome architecture: DNA in prokaryotes or in organelles usually forms a loop – a *circular* chromosome; on the other hand, chromosomes in the nuclear genomes of eukaryotes are typically *linear*, ending with a structure called *telomere* on each side.

Mutations. The DNA replication process is not infallible and from time to time, a "typo" occurs. These typos are called *point mutations* and are one of the sources of genetic variation. From time to time, even larger errors occur. If we explore, for example, human genome, we can find a lot of repeating sequences. These are the results of *duplications*, which copy a segment of DNA. Common are *tandem duplications*, in which the copied segment is inserted next to the original (see Fig. 1.5(a)). General duplications are also called *retrotranspositions* (Fig. 1.5(b)). Similarly, a segment of DNA may be deleted (Fig. 1.5(c)).



(a) Tandem duplication.

(b) Duplication.

(c) Deletion.

(d) Whole genome duplication.

Figure 1.5: Large scale mutations. Each arrow represents a segment of DNA; recall that DNA consists of two strands running in opposite direction.

An extreme case is a *whole genome duplication* (Fig. 1.5(d)), which may occur due to abnormal cell division. Most organisms, including humans, are *diploid*, i.e., they have two sets of chromosomes – each inherited from one parent. But there are species, especially plants, which are *polyploid* – they underwent a whole genome duplication and thus have more than two sets of chromosomes. For example, there are strains of wheat which are diploid, tetraploid (macaroni wheat), and hexaploid (bread wheat) – having respectively 2, 4, or 6 sets of chromosomes (Simmonds et al., 1976).

Rearrangements. If we compare genomes of *different* species, we often find very similar segments of DNA, since the two species share a common ancestor and important segments of DNA, such as genes, are usually well conserved. This is because a deleterious mutation may disable a gene. If the gene encoded some protein, the protein may not be produced. This may mean that some process in a cell fails and this may have even lethal consequences. The organism may have a lower chance to mate and thus, such a mutation may not proliferate to the descendants of the organism. On the other hand, neutral or beneficial mutations may accumulate throughout the evolution.

Let us take for example human and mouse genome and choose a different colour for each mouse chromosome. If we paint each segment of human DNA similar (in sequence) to a mouse DNA by the colour of the corresponding chromosome in mouse, we obtain Fig. 1.6. We can see that there are long segments of DNA, called *conserved syntenies*, that are well-preserved, but *shuffled* around the genome. This is the result of various rearrangement mutations depicted in Fig. 1.8.

Another example is shown in Fig. 1.7: If we compare the mitochondrial genome of cabbage and turnip, we can identify five segments of DNA which are sequentially almost identical, however, their order in cabbage and in turnip is different. Figure 1.7 also shows three reversals which might have occurred

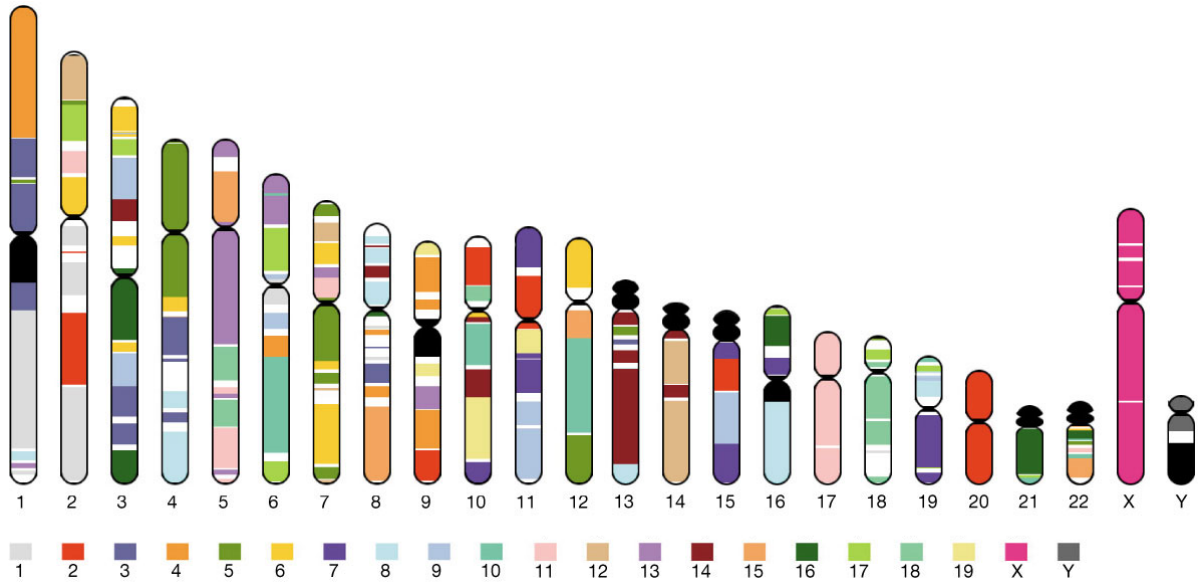


Figure 1.6: Human chromosomes with segments containing at least two genes whose order is conserved in the mouse genome as colour blocks. Each colour corresponds to a particular mouse chromosome. *Source:* International Human Genome Sequencing Consortium, Lander et al. (2001).

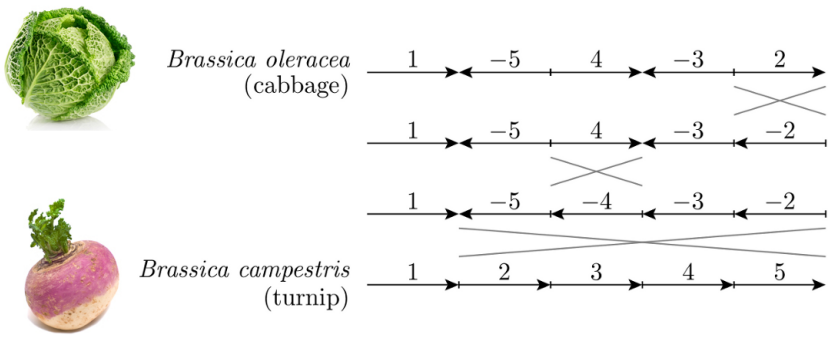


Figure 1.7: Mitochondrial genomes of cabbage and turnip. Let us number the conserved segments $1, \ldots, 5$ and depict them as arrows; a minus sign and an arrow in opposite direction represent a segment on the opposite strand. The 'X' marks show segments which were probably reversed during the evolution.

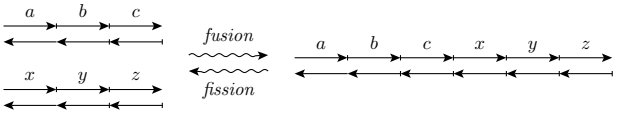
during the evolution of cabbage and turnip from their common ancestor and which provide one possible explanation for these data.

The most common rearrangement mutation is a *reversal* (also called *inversion*, see Fig. 1.8(a)), which may occur when the double helix breaks at two points. The cell machinery tries to repair this defect, but accidentally, it "glues" the middle part in the opposite direction. A different mechanism is shown in Fig. 1.9. Here, a motif gets accidentally paired with its remote copy on the other strand (a so called *crossover*). Resolving this intertwining leads to reversing the segment between the two motifs.

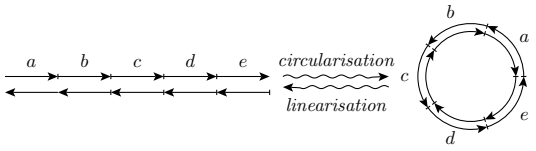
(a) Reversal. The order of markers b, c, d is reversed. Moreover, they also move onto the opposite strand, so their orientation is also flipped.

(b) Transposition. Block b, c is moved after block d. Note that this is the same as if block d moved before b, c. In other words, two consecutive blocks are swapped.

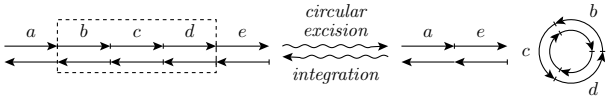
(c) Translocation. Arms of two chromosomes are interchanged.



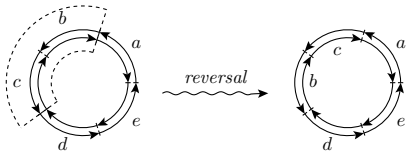
(d) Fusion of two chromosomes and the reverse process – chromosomal fission. (This can be treated as an extreme case of translocation, where an empty block is exchanged for a whole chromosome.)



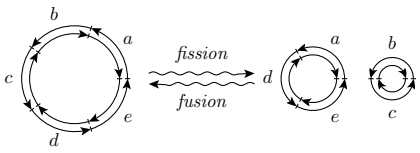
(e) Circularization and linearization of a chromosome. (This can be treated as a special case of circular excision/incorporation.)



(f) Circular excision and the reverse process – integration of a circular chromosome.



(g) Reversal in a circular chromosome.



(h) Fusion and fission of circular chromosomes.

Figure 1.8: Rearrangements changing the order of genes, number, or architecture of chromosomes. These operations, however, do not change the gene content.

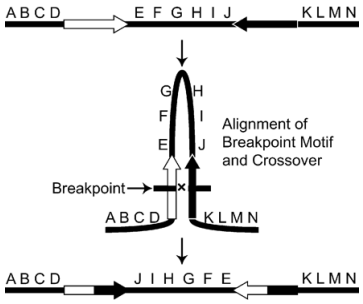


Figure 1.9: One possible mechanism of reversal: The white and black arrows represent similar sequences which are accidentally paired. The sequence inbetween is reversed when the crossover is resolved.

Similarly, if the chromosome breaks at 3 places and the pieces are restored in the wrong order, we end up with a *transposition* (Fig. 1.8(b)). Alternatively, there are sequences of DNA, called *transposons*, which can move (transpose) to other positions in the genome by themselves.

If the genome consists of several chromosomes, two different chromosomes may break, or a crossover between two different chromosomes may occur, which leads to a *translocation* (see Fig. 1.8(c)).

Karyotype, the number and appearance of chromosomes, may change by *fusion* of two chromosomes or *fission* of a single chromosome (Fig. 1.8(d)). A linear chromosome may turn into a circular and vice versa (Fig. 1.8(e)). A circular segment may be excised (Fig. 1.8(f)) and later reincorporated into a linear chromosome (this is actually one of the mechanisms of transposition).

Finally, similar rearrangements may occur in circular genomes (Fig. 1.8(g) and 1.8(h)).

Problems motivated by comparative genomics. In this thesis, we will be interested mainly in rearrangements, i.e., large-scale mutations that do not change the gene content, but shuffle the genes around the genome.

We can compare gene orders of different species and estimate how similar they are by counting the number of rearrangements needed to transform one genome into another. See for example Fig. 1.10 depicting mitochondrial genomes of five yeast species from the genus *Candida*. In each genome, there are 14 protein coding genes and two ribosomal RNA genes (plus some tRNA genes, not shown). Even though these species are quite close, we can see a variety of genome architectures: *C. subhashii* and *C. parapsilosis* are linear, *C. frijolesensis* has two linear chromosomes, and *C. jiufengensis* and *C. tropicalis* have circular chromosomes.

The following questions arise naturally while one studies these species:

- Which two *Candida* species are the most closely related?
- How did the ancestor of *C. parapsilosis* and *C. jiufengensis* look like? What was its purported gene order and how did it evolve into *C. parapsilosis* and *C. jiufengensis*?
- The same question may be asked for the ancestor of *C. tropicalis* and *C.frijolesensis*.

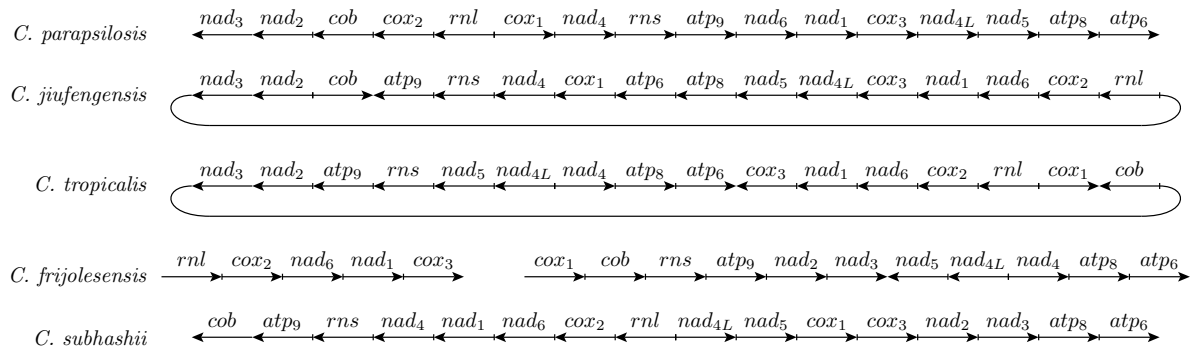


Figure 1.10: Gene orders of mitochondrial DNA of five *Candida* species.

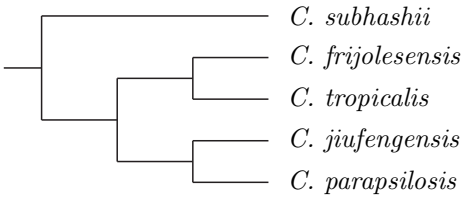


Figure 1.11: Phylogenetic tree of the five *Candida* species.

- More generally, assume that the correct phylogenetic relationships between the given five species are as depicted in Fig. 1.11. How did the gene orders of all ancestral species look like?
- Even more generally, taking into account the gene orders, which one of all 105 phylogenetic trees is the most likely?

In fact, all of these questions can be simply summarized as

What happened during the evolution?

Why to study rearrangements. From the biological point of view, the rearrangement mutations are interesting, because they may cause inability of organisms to cross-breed and thus emergence of new species (a so called *speciation*).

Questions like what is the evolutionary distance between the two species and which is the correct evolutionary tree are also studied at the level of DNA sequence. Here, the measure of similarity is the sequence similarity (the number of point mutations). The advantage of rearrangements is that these events are much rarer in the evolution, so they allow us to look deeper into the evolutionary history.

1.3 Computational Problems in Genome Rearrangements

Probabilistic methods vs. parsimony. There are basically two approaches used to answer questions about evolution. The first one is *probabilistic*: we model the evolutionary events as random processes, and we either ask which values of the model parameters maximize the probability that we obtain the

observed data (the maximum likelihood methods), or compute an approximate probability distribution over the parameters of interest by sampling from the *a posteriori* distribution given the observed data (Bayesian inference).

The second approach is based on the *parsimony principle* – the most succinct explanation is considered the best. More specifically: An evolutionary scenario best explains gene orders of the extant species, if it has the minimum number of rearrangements.

Even though the probabilistic approach would be more satisfactory, it is the parsimony principle that is prevailing in the field of genome rearrangements and that we use in this thesis. There are basically three reasons for this:

1. Firstly (on the positive side), the rearrangement mutations are very rare so it makes perfect sense to prefer solutions which predict as few mutations as possible.

2. Secondly, (on the negative side), we do not have a decent probabilistic model. We know that among the rearrangement operations, reversals and translocations are the most common. However, we have no idea, what is the relative frequency of reversals compared to, say, translocations or transpositions. Furthermore, even if we consider reversals only: What is the relative frequency of reversals of different lengths? Perhaps, shorter reversals are more frequent than longer ones. Finally, some places in the genome are more prone to breakage than others and these places have higher probability of being an endpoint of a reversed interval. A good probabilistic model for genome rearrangements should take all these factors into account.

3. Thirdly, the rearrangement problems in the parsimony setting are interesting (and hard) enough to be studied in their own right.

Genome models. In this chapter, we introduce the fundamental problems in the field of genome rearrangements. Naturally, the answers to these problems depend on what do we mean by "genome" and what do we mean by "rearrangement". Do we work with linear genomes only, or do we consider also circular ones? What are the operations that rearrange the genomes throughout the evolution? Reversals? Translocations? Transpositions? Fusions and fissions? Some combination of the former?

For example, if all the species in consideration have multiple linear chromosomes, also the ancestral genomes should be multilinear and the most common rearrangement operations are reversals and translocations. On the other hand, if all the species in consideration have only a single circular chromosome, perhaps the ancestral genomes should also be circular and the only allowed rearrangement operation should be reversal.

Various genome models have been proposed. These may be divided into *signed* and *unsigned* (depending on whether we know the orientation of genes), *linear*, *circular*, and *mixed* (depending on what chromosomal architectures are allowed), and *unichromosomal* and *multichromosomal* (depending on whether multiple chromosomes are allowed).

Different genome models will be discussed in Chapter 2. However, for each genome model, we can study essentially the same problems. Thus, for now, suppose that we have settled for a particular genome model and let us introduce different kinds of problems that we meet in the field of genome rearrangements.

Rearrangement distance. The biological motivation for our first problem is in Fig. 1.12. We imagine species α, an extinct common ancestor of two extant species π and γ. Starting with the common ancestral gene order, after speciation the two species evolve independently and undergo mutations. We can determine the gene orders of π and γ and we can ask, how many mutations occurred during the evolution. Or, according to the parsimony principle (since rearrangement mutations are rare), what is the *minimum* number of mutations necessary to explain given gene orders? Since all genome models considered here are symmetric, we can rephrase that as: What is the minimum number of rearrangements needed to transform π into γ?

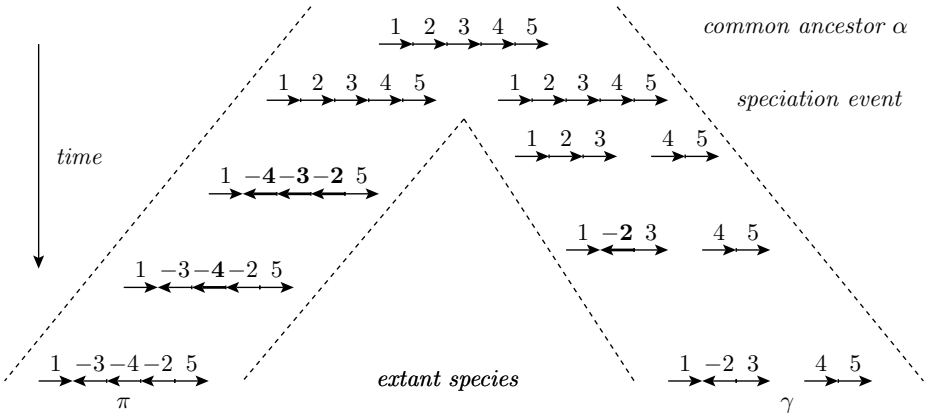


Figure 1.12: Evolution of gene orders π and γ from the common ancestor α.

Problem 1 (Rearrangement distance). *Given two genomes π and γ, find the distance $d(\pi, \gamma)$, the minimum number of rearrangements needed to transform π into γ.*

The genomic distance between π and γ is a lower bound on the true number of mutations throughout the evolution of π and γ from their common ancestor α. It is a fundamental problem in genome rearrangements, since rearrangement distance is indeed a distance measure (mathematically speaking, the set of all gene orders is a metric space), and all of the other rearrangement problems build upon the ability to compute this distance.

A related problem is the *sorting problem*, in which we not only want to compute the genomic distance, but also one particular shortest sequence of rearrangement operations transforming π into γ.

Problem 2 (Sorting). *Given two genomes π and γ, find a shortest sequence of rearrangements transforming π into γ.*

Note that computing the rearrangement distance and finding one optimal sorting scenario are two distinct problems, the sorting problem being possibly harder. If we can sort efficiently, we can also

compute the distance efficiently. On the other hand, in all models that we consider in this thesis, there are only polynomially many rearrangements available in each step, and the distance is at most linear in the number of genes. Thus, if we can compute the distance in polynomial time, we can also sort in polynomial time by trying all possible moves and searching for one which decreases the distance. Usually, there are more efficient ways to sort the genome and we will study them together with computing the genomic distance in the next chapter.

Median. Note that the gene orders π and γ do not provide enough information to reconstruct the ancestral genome α. Therefore, let us consider gene order ρ of an outgroup species. For instance, returning to our example with yeast gene orders and the phylogenetic tree in Fig. 1.11, if we wanted to reconstruct, say, the gene order of *C. parapsilosis* and *C. tropicalis* common ancestor, we could take *C. subhashii* as an outgroup.

According to the parsimony principle, the answer we consider the best is a gene order M which minimizes the sum of pairwise distances between M and π, γ, and ρ. Such a genome is called *median* and the corresponding problem is called *median problem.*

Problem 3 (Median). *Given three genomes π, γ, ρ, find genome M, called* median*, that minimizes the sum of distances from the three given genomes $d(M, \pi) + d(M, \gamma) + d(M, \rho)$.*

Unfortunately, as we will see in Chapter 3, for most genomic distances this problem is hard. However, we will describe several practical algorithms for computing the median.

Rearrangement phylogenies. As mentioned in the previous section, even more ambitious problem is to compute gene orders of all the ancestral species on a given phylogeny – a so called *small phylogeny problem.*

Problem 4 (Small phylogeny). *Given a phylogenetic tree and genomes of the extant species (leaves of the tree), find genomes of the ancestral species (internal vertices), while minimizing the overall number of rearrangements throughout the evolution (the sum of distances along the edges of the tree).*

If the phylogenetic tree of the species is not given, we may try to reconstruct one based on the gene orders. This is the *large phylogeny problem.*

Problem 5 (Large phylogeny). *Given the genomes of extant species, find both a phylogenetic tree and genomes of the ancestral species, while minimizing the number of rearrangements in the evolutionary history.*

We will study practical solutions to both small and large phylogeny problems in Chapters 5 and 7.

Whole genome duplication. The problems above are usually studied for genomes having no duplications. We will see in Section 2.5, that once we start considering duplications, even computing the

distance becomes hard. There is, however, a very special case of duplication which deserves more study: a whole genome duplication.

Genome halving. Imagine a genome that underwent a whole genome duplication and then evolved by large-scale rearrangements (Fig. 1.13). Even though immediately after the duplication the genome had two perfect copies of each chromosome, nowadays we observe the copies scattered all over the genome. The goal of the *halving problem* is, given a present genome with two copies of each gene, to reconstruct the genome immediately after the duplication. In other words, the goal of genome halving is to find a *perfectly duplicated* genome with the smallest genomic distance from the given genome.

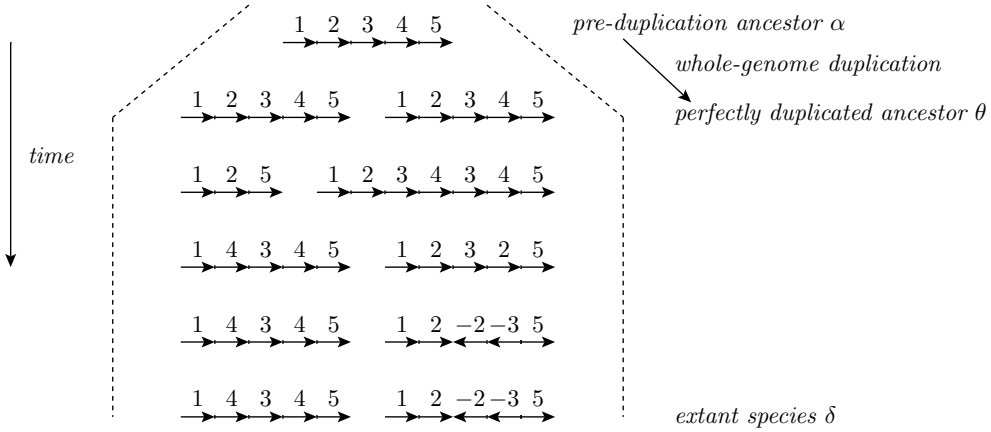


Figure 1.13: The ancestral genome undergoes a whole genome duplication and subsequently evolves by rearrangements. The goal of genome halving is to reconstruct the genome immediately after the whole genome duplication, given the gene order of the extant species.

Problem 6 (Genome Halving). *Given a duplicated genome δ, find a perfectly duplicated genome θ that minimizes the rearrangement distance $d(\theta, \delta)$.*

We will study the halving problem and its variants in Chapter 4.

1.4 Outline of the Thesis and Contributions

This thesis consists of two parts. In the first part, we survey prior research on the rearrangement problems:

- In Chapter 2, we introduce different genome models and study the most basic problems in genome rearrangements – computing the distance and sorting. The chapter is centered around the most general, double cut and join model, and other models (reversal and reversal-translocation models) are viewed as its restrictions.
- Chapter 3 deals with the problem of reconstructing a single ancestral gene order (Problem 3).

Unfortunately, the problem is hard for most genomic distances, but we will describe practical ways of solving it.

- In Chapter 4, we study the halving problem and its variants such as double distance, guided halving, and genome aliquoting. In these problems, we try to reconstruct the pre-duplication state of a genome that underwent a whole genome duplication. These problems are usually approached with the techniques introduced in the previous two chapters.

- Finally, in Chapter 5, we survey the attempts to solve biologically most interesting problems in rearrangements which also happen to be computationally hardest: reconstructing the phylogenetic tree and the evolutionary history on the set of many species.

The second part of the thesis contains our contributions:

- In Chapter 6, we define a new model, a restricted version of the double cut and join model from Section 2.2, and in this model, we study the three classical problems: sorting, halving, and median.

  We propose a new $O(n \log n)$ algorithm for the restricted sorting problem, thus improving on the known quadratic time algorithm. We solve the restricted halving problem and give an algorithm that computes a multilinear halved genome in linear time. Finally, we show that the restricted median problem is NP-hard as conjectured.

- Chapter 7 is more practical, and deals with the problem of reconstructing ancestral gene-orders. We propose a new approach to solving the small phylogeny problem and implement it in our software `PIVO`. We demonstrate the accuracy of our program on the well-studied dataset of *Campanulaceae* chloroplast genomes, and apply it to the reconstruction of rearrangement histories of newly sequenced mitochondrial genomes of pathogenic yeasts from *Hemiascomycetes* clade.

- In the last chapter, we solve several open problems concerning computational complexity of rearrangement problems in the breakpoint model. We give an $O(n\sqrt{n})$ algorithm for the median problem improving on the cubic algorithm by Tannier et al. (2009). Moreover, we show that the problem is equivalent (under linear reduction) to finding maximum matching. Thus, any improvement to our solution would imply a better algorithm for the maximum matching, which has been an open problem for more than 30 years (Micali and Vazirani, 1980).

  The general breakpoint model is one of the very few models, where the median problem is easy. It was an open problem whether the results of Tannier et al. (2009) can be generalized to reconstructing evolutionary history of many species. We prove that the more general small phylogeny problem is NP-hard. Surprisingly, we show that it is already NP-hard (or even APX-hard) for four species (a quartet phylogeny). In other words, while finding an ancestor for three species is easy, finding two ancestors for four species is already hard.

Part I

Survey

Chapter 2

Distances Between Genomes

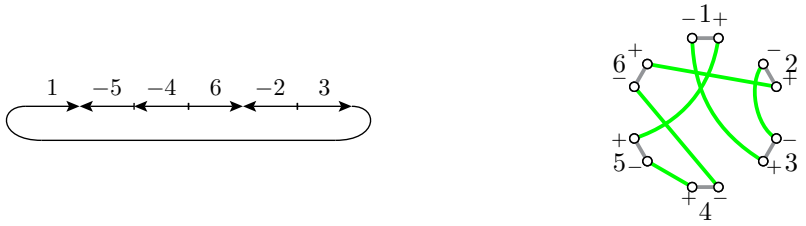
There are dozens of genome models, depending on what combination of karyotype and subset of rearrangement operations we choose. However, not all of them are biologically interesting. We will be mainly interested in signed genomes, where the orientation of all genes is known.

In this chapter, we review the most important genomic distances studied in the literature. We start in Section 2.1 by formalizing the notion of a genome and by introducing the simplest distance measure, the breakpoint distance. In Section 2.2, we define a very simple and general operation, called double cut and join (DCJ). With this operation, we can model all the interesting rearrangement operations described in the previous chapter. We will then present the last two models as restrictions of the DCJ model, reusing the framework and achieved results. In Section 2.3, we study distance and sorting problems for genomes with single chromosome evolving by the most common rearrangement operation – reversal, and in Section 2.4, we generalize the results to multilinear chromosomes, where translocations, fusions and fissions are also common. Finally, in Section 2.5, we briefly mention other distances and point to further literature.

2.1 Genome Representation and Breakpoint Distance

The units that are shuffled around the genome may be individual genes or even larger blocks called conserved syntenies. We call them simply *genes* or *markers*. We will assume that every genome consists of the same set of markers, usually numbered $1, 2, \ldots, n$.

Unsigned genomes. If we do not know the orientation of markers, we model genomes as *unsigned* genomes and we represent them by undirected graphs with maximum degree 2. Each marker is represented by a single vertex and consecutive markers are connected by edges called *adjacencies*. The components of such graphs are paths and cycles, which naturally correspond to linear and circular chromosomes, respectively.



(a) The order of genes in a genome. Each arrow corresponds to a single marker with known orientation.

(b) Representation of the genome on the left by a perfect matching. The green edges are the adjacencies of π, the gray edges form the base matching B. The Hamiltonian cycle $\pi \cup B$ corresponds to the single circular chromosome.

Figure 2.1: Example of a circular genome π and its representation by a perfect matching.

Signed genomes. If we do know the orientation of markers, we represent each marker by two vertices. These are called *extremities* of the marker and they correspond to the left and right side, or *tail* and *head* of the marker. If m is a marker, we denote m^- the left side and m^+ the right side. If two extremities p and q lie next to each other in the genome, we say that the pair $\{p, q\}$, written simply pq, is an *adjacency*. For example, in Fig. 2.1(a), head of marker 1 is next to head of marker 5, so 1^+5^+ is an adjacency. Similarly, tail of marker 5 is next to head of marker 4, so 5^-4^+ is an adjacency. All the adjacencies of the genome in Fig. 2.1(a) are depicted in Fig. 2.1(b) by green edges.

Definition 1. *A* general (multichromosomal circular) signed genome *is a set of adjacencies such that each extremity belongs to exactly one adjacency. In other words, a genome is a perfect matching of the extremities.*

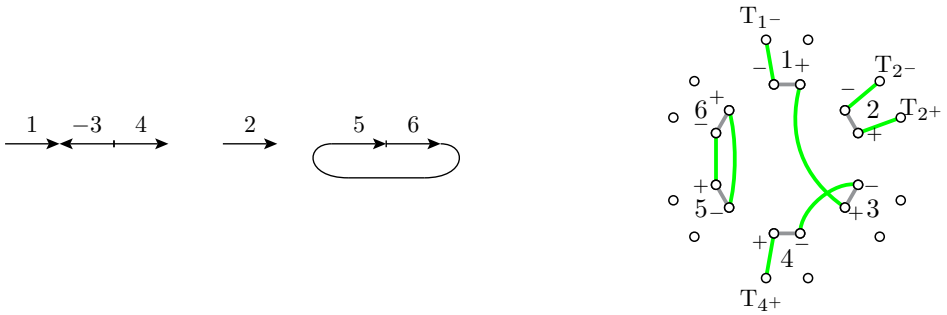
Let us define an auxiliary *base matching* B consisting of the *marker edges* m^+m^- for each marker m (the gray matching in Fig. 2.1(b)). Then all vertices have degree 2 in the union[1] $\pi \cup B$, and $\pi \cup B$ decomposes into a set of cycles, which naturally correspond to the circular chromosomes of our genome.

In the *general* (multichromosomal circular) model, genomes can have multiple circular chromosomes, and any perfect matching π corresponds to a genome. In the *unichromosomal circular* model, we require that the genome only consists of a single chromosome, so $\pi \cup B$ is a Hamiltonian cycle as in Fig. 2.1(b). Such a matching π is sometimes called a Hamiltonian matching.

Representing linear chromosomes. There are two ways of generalizing the model to include linear chromosomes. In the first one, we add a special vertex T_x called *telomere* for each extremity x. If x lies at the end of a linear chromosome, we say that $x\mathrm{T}_x$ is a *telomeric adjacency*.

Definition 2. *A* general (mixed) signed genome *is a set of adjacencies such that each extremity belongs to exactly one adjacency. In other words, a genome is a matching on the set of extremities and telomeres such that each telomere* T_x *may only be matched with* x *and all extremities are matched.*

[1] technically, this is a disjoint or multiset union; we allow parallel edges forming 2-cycles



(a) A genome with two linear and one circular chromosome.

(b) The same genome represented as a set of adjacencies (green matching). Gray edges form the base matching B. Components of $\pi \cup B$ are paths and cycles, corresponding to linear and circular chromosomes, respectively.

Figure 2.2: Example of a mixed genome π and its representation.

In this model, $\pi \cup B$ consists of cycles and paths ending at telomeres, which naturally correspond to the circular and linear chromosomes of our genome, respectively (see Fig. 2.2). In the *multilinear* model, we require that all components of $\pi \cup B$ are paths and in the (unichromosomal) *linear* model, we require that $\pi \cup B$ is a single path going through all extremities.

Alternatively, sometimes it is more convenient to represent the telomeres by a single vertex T. A genome is again defined as a set of adjacencies such that each extremity belongs to exactly one adjacency.

Notation. Naturally, when writing out the genomes, we use a linear notation and we simply list the markers along the chromosomes. For linear chromosomes, we choose a direction and then list the markers from left to right; we write $\overrightarrow{g}$, if extremity g^- is before g^+ and $\overleftarrow{g}$ otherwise. Similarly, for circular chromosomes, we choose a starting point and direction in which we list the markers. We use parentheses for linear chromosomes and square brackets for circular chromosomes. Thus, we would write the genome on Fig. 2.2(a) as $(\overrightarrow{1}, \overleftarrow{3}, \overrightarrow{4}), (\overrightarrow{2}), [\overrightarrow{5}, \overrightarrow{6}]$. Note that the representation is not unique and we could also write the first chromosome as $(\overleftarrow{4}, \overrightarrow{3}, \overleftarrow{1})$ and the circular chromosome as $[\overleftarrow{6}, \overleftarrow{5}]$ or $[\overleftarrow{5}, \overleftarrow{6}]$.

We will write $-g$ for the reversed marker g, i.e., $-\overleftarrow{g} = \overrightarrow{g}$ and $-\overrightarrow{g} = \overleftarrow{g}$. If $I = m_1, \ldots, m_k$ is a sequence of markers, then $-I = -m_k, \ldots, -m_1$ is the reversed sequence. For example, if C is a chromosome, then $C = -C$.

A short digression on linear and circular genomes. Other equivalent ways of defining genomes can be found in the literature: Unsigned linear genomes naturally correspond to classical permutations. Signed linear and circular genomes can be defined using permutations, however, defining more general models this way becomes cumbersome (this is why we prefer the opposite direction starting with the most general definition using graphs and then defining linear, circular, or multilinear genomes by restricting the components of these graphs).

If we worked with linear genomes only, we could have defined them as signed permutations. Formally,

a *signed permutation* on $\{1, 2, \ldots, n\}$ is a permutation π of the set $\{-n, \ldots, -2, -1, 1, 2, \ldots, n\}$ such that $\pi_{-i} = -\pi_i$. Thus, the whole mapping is specified by the mapping of the positive elements (which determine the order and orientation of all genes).

Note that both signed permutations and paths in the graph-theoretic formulation are unchanged when the whole chromosome is reversed. On the other hand, when working with linear genomes, it is sometimes preferable to work with *linear extensions* of the permutations: we extend a signed permutation $\pi = (\pi_1, \ldots, \pi_n)$ by adding sentinel markers $\overrightarrow{0}$ at the beginning and $\overrightarrow{n+1}$ at the end. This way, we can fix the orientation of chromosomes.

Circular genomes could be defined by calling signed permutations π and γ equivalent if π can be obtained from γ by rotating the elements and possibly reversing the whole set of elements, flipping all signs. A *genomic circular permutation* is then an equivalence class under this equivalence relation.

Breakpoint distance for linear genomes. Probably the simplest distance measure between two permutations is the breakpoint distance introduced by Sankoff and Blanchette (1997). As we already mentioned in Chapter 1, we can rename and flip the markers in both permutations so that one of the permutations is the identity.

Let π be a linear extension of a signed permutation; each consecutive pair (π_i, π_{i+1}) is called a *point*, also written as $\pi_i \bullet \pi_{i+1}$. If a point is of the form $\overrightarrow{k} \bullet \overrightarrow{k+1}$ or $\overleftarrow{k+1} \bullet \overleftarrow{k}$, we call it a *common adjacency*, otherwise, it is a *breakpoint*. In other words, breakpoints are positions, where π has to be broken in order to transform it into the identity permutation.

Let us denote by $bp(\pi)$ the number of breakpoints in permutation π, the *breakpoint distance* from the identity permutation. In general, the breakpoint distance between two permutations π and γ is defined as

$$bp(\pi, \gamma) = bp(\gamma^{-1} \circ \pi).$$

Even though there are no underlying rearrangement operations in the breakpoint model, we can transform π into γ by first cutting π into $bp(\pi, \gamma) + 1$ pieces and then rejoining the pieces in the correct order. Thus, we could interpret $bp(\pi, \gamma)$ as the number of fissions and fusions needed to transform π into γ.

Various rearrangement operations generally increase the breakpoint distance between the genomes, unless they reuse already created breakpoints. Thus, the breakpoint distance may serve as a simple lower bound for other distances. For example, a single reversal can only create at most two breakpoints, so the reversal distance (studied in Section 2.3) is at least half the breakpoint distance.

Breakpoint distance for general genomes. In the general setting, when computing the breakpoint distance between multichromosomal genomes π and γ, we simply look at the graphs of the two given genomes and count how many adjacencies (and telomeres) do they have in common. Tannier et al. (2009)

advocate the following definition of the breakpoint distance:

$$bp(\pi, \gamma) = n - a(\pi, \gamma) - \frac{e(\pi, \gamma)}{2},$$

where $a(\pi, \gamma)$ is the number of adjacencies that π and γ have in common, and $e(\pi, \gamma)$ is the number of telomeres that they have in common. Note that this distance does not have to be integral, but for unichromosomal genomes, it coincides with the definition in the previous paragraph if we consider linear extensions of the genomes. In this definition, fusions and fissions can create at most one breakpoint, whereas reversals or translocations at most two.

2.2 The Double Cut and Join Distance

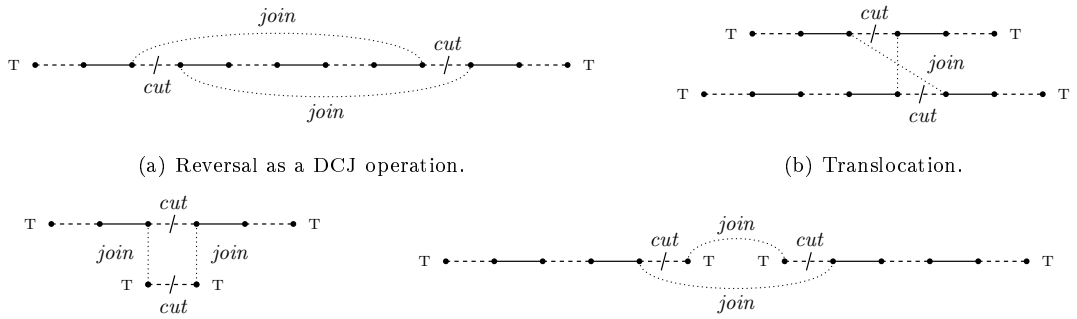
The double cut and join (DCJ) model was introduced by Yancopoulos et al. (2005) and revised by Bergeron et al. (2006b). It models signed genomes, possibly with multiple linear or circular chromosomes which evolve by double cut and join operations.

DCJ operation. A double cut and join operation models in a unified way all the rearrangement operations introduced in Section 1.2. In a single DCJ operation, we break the genome at two points and then rejoin the four created endpoints in a different way. More formally: A *double cut and join operation acting on adjacencies pq and rs* replaces them by either adjacencies pr, qs, or ps, qr. We say that the operation *cuts* pq and rs and *joins* either pr, qs, or ps, qr. For simplicity, formally we assume that genomes contain infinite number of loops at the telomeric vertex T (we may think of the loops as empty chromosomes TT). In the definition of a DCJ operation, the adjacencies pq and rs that we cut may be telomeric or even a telomeric loop TT.

By DCJ operations, we can mimic every common rearrangement operation in genomes: For example, to reverse an interval, we cut at its boundaries and join the endpoints as in Fig. 2.3(a). If we cut and join adjacencies of different linear chromosomes, we get a translocation (Fig. 2.3(b)). We can fuse two chromosomes into one by cutting their telomeric adjacencies pT and qT and joining pq and TT (as a byproduct, we get an empty chromosome TT; Fig. 2.3(d)) or fission one chromosome by cutting adjacencies pq and TT and joining pT and qT (see Fig. 2.3(c)).

Other rearrangements that can be simulated by DCJ operations are linearization and circularization of a chromosome, circular excision and incorporation, or fusion and fission of circular chromosomes.

Furthermore, as we already mentioned in Section 1.2, transposition can be modeled by excision of a circular fragment and its reincorporation at another location. In fact, excision and reincorporation can mimic a more general operation called *block interchange* (see Fig. 2.4). Transposition is a special case of block interchange, where the two interchanged blocks are adjacent (Fig. 1.8(b) in Section 1.2). Transposition (or reversed transposition) occurs when the temporary circular chromosome breaks at the same place where it was joined during excision. This is often the case since some positions of DNA are more prone to breakage than the others. In fact, the block interchange operation as a generalization of



(a) Reversal as a DCJ operation.

(b) Translocation.

(c) Fission; we cut a telomeric loop.

(d) Fusion is a special case of a translocation that creates an empty chromosome TT.

Figure 2.3: Examples of operations in the DCJ model. All the nodes marked with letter T correspond to the single telomeric vertex.

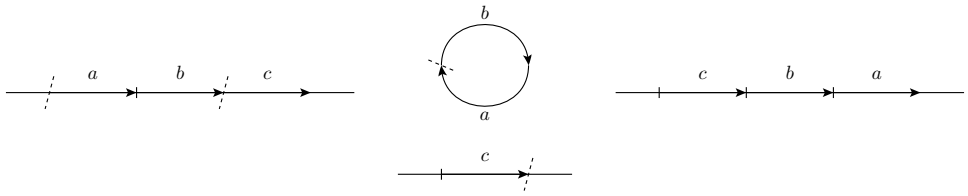


Figure 2.4: To interchange blocks a and c (left) in the DCJ model, we cut before a and after b and create a temporary circular chromosome (center). The next operation cuts between a and b and after c and reincorporates the blocks in the correct order (right).

transposition is not very biologically plausible. It was introduced by computer scientists, since it greatly simplifies problems; for example, sorting by block interchanges is easy (Christie, 1996), while sorting by transpositions only is NP-hard (Bulteau et al., 2012b).

DCJ distance and scenarios. Naturally, a sequence of k DCJ operations transforming genome π into γ is called a *DCJ scenario of length* k. A scenario of minimum length is called *optimal* and its length is the *DCJ distance between* π *and* γ, denoted $dcj(\pi, \gamma)$. More generally, a sequence of k DCJ operations transforming π into π' is *optimal* with respect to γ, if $dcj(\pi, \gamma) = dcj(\pi', \gamma) + k$, i.e., all the operations transform the genome towards γ.

There is an alternative interpretation of the DCJ distance – we can view the DCJ model as a model with weighted operations:

- reversals, translocations, fusions, and fissions have weight 1,
- translocations and block interchanges have weight 2.

It can be proved that there is always an optimal scenario using only reversals, translocations, and block interchanges and any DCJ scenario with minimum weight is optimal. We will study such scenarios in Chapter 6.

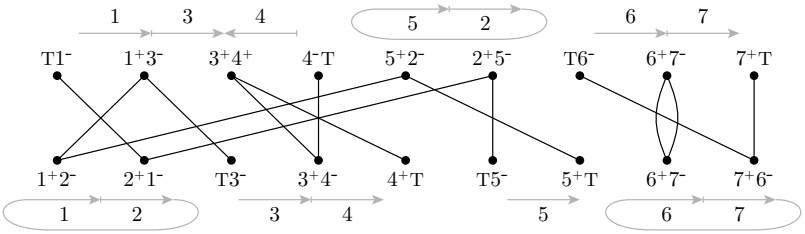


Figure 2.5: Adjacency graph for the two genomes. It consists of one cycle, two odd-length paths, and two even-length paths. Thus the distance between these genomes is 5.

Computing the DCJ distance. The distance and a sorting scenario can be calculated using an adjacency graph $AG(\pi, \gamma)$. This is a bipartite multigraph, where vertices are adjacencies of π and γ; an adjacency in π is connected with an adjacency in γ by one edge for each extremity that they share (Fig. 2.5). Since every adjacency is connected with one (telomeric) or two other adjacencies, this graph consists of paths and cycles only. If π and γ share a *common adjacency*, this corresponds to a cycle of length 2 or path of length 1 (common telomere) in the adjacency graph. Note that when π and γ are equal, their adjacency graph consists of 2-cycles and 1-paths only. Thus, we may think of transforming π into γ as breaking cycles and paths in the adjacency graph $AG(\pi, \gamma)$.

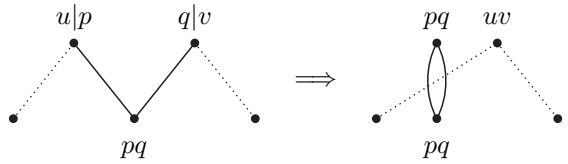
The following observations are easily proved (Bergeron et al., 2006b):

Lemma 1. *Let π and γ be two genomes on n markers, let c be the number of cycles and p_o the number of paths of odd length in the adjacency graph $AG(\pi, \gamma)$. Then $\pi = \gamma$ if and only if $n - (c + p_o/2) = 0$.*

Lemma 2. *The application of a single DCJ operation changes either the number of cycles in the adjacency graph by $+1$, 0, or -1 or the number of odd-length paths by $+2$, 0, or -2.*

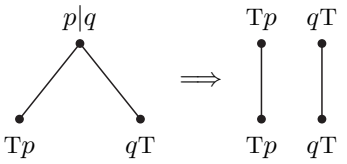
Corollary 1. *Let π and γ be two genomes on n markers, let c be the number of cycles and p_o the number of paths of odd length in the adjacency graph $AG(\pi, \gamma)$. Then $dcj(\pi, \gamma) \geq n - (c + p_o/2)$.*

Thus, we have a lower bound on the number of DCJ operations needed to transform one genome into another. On the other hand, a DCJ operation is so general that in any situation there is an appropriate optimal operation: Let pq be an adjacency in γ that is not present in π; say π contains adjacencies pu, qv (u, v may also be telomeres). Then by one DCJ operation cutting pu and qv, and joining pq and uv, we create a new 2-cycle:



The remaining structure, either a path or a cycle, which contained *up–pq–qv* is just shortened by two edges. Thus, the number of cycles increases and the distance is decreased by 1.

Once all the adjacencies of γ belong to 2-cycles, the only undesired components in the adjacency graph are paths of length 2 (where the telomeric adjacencies belong to γ). These correspond to chromosomes which have to undergo fusion/fission in order to complete the transformation. By one DCJ operation, we can either break a 2-path into two 1-paths, or join the ends to form a 2-cycle. In each case the distance decreases by 1.

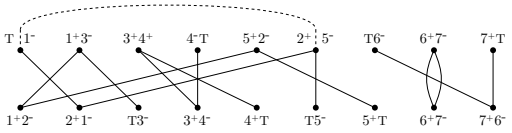


Thus the lower bound can always be met and we can formulate a theorem giving the DCJ distance between two genomes:

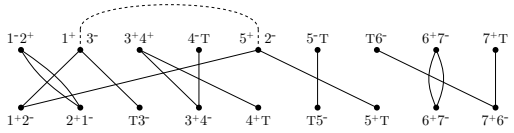
Theorem 1 (Bergeron et al. (2006b)). *Given two genomes π and γ on n markers, let c be the number of cycles and p_o be the number of paths of odd length in the adjacency graph $AG(\pi, \gamma)$. Then the distance between π and γ is*

$$dcj(\pi, \gamma) = n - (c + p_o/2).$$

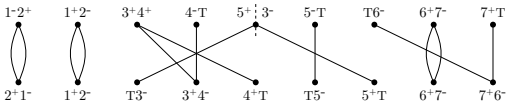
Moreover, from the discussion it should be clear that we can generate a particular sorting scenario in linear time. One example of DCJ sorting is shown in Fig. 2.6.



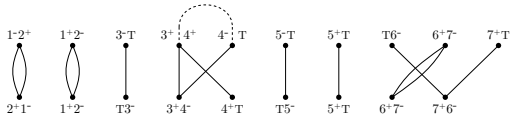
(a) Adjacency graph of the input genomes. Step 1: Cut $\mathrm{T}1^-$, 2^+5^- and join $\mathrm{T}5^-$, 1^-2^+.



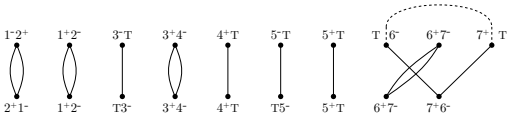
(b) Step 2: Cut 1^+3^-, 5^+2^- and join 1^+2^-, 5^+3^-.



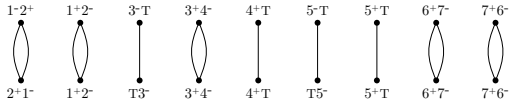
(c) Step 3: Cut 5^+3^- (and TT) and join $5^+\mathrm{T}$, $3^-\mathrm{T}$.



(d) Step 4: Cut 3^+4^-, $4^-\mathrm{T}$ and join 3^+4^-, $4^+\mathrm{T}$.



(e) Step 5: Cut $\mathrm{T}6^-$, $7^+\mathrm{T}$ and join 6^+7^- (and TT).



(f) The adjacency graph now consists only of 2-cycles and 1-paths so the two genomes are equal.

Figure 2.6: Example of DCJ sorting. We transform the top genome from Fig. 2.5 into the bottom one using 5 DCJs.

Table 2.1: History of results on the sorting by reversals problem.

	Distance	Sorting	Notes
Hannenhalli and Pevzner (1995)	$O(n^2)$	$O(n^4)$	first polynomial algorithm
Berman and Hannenhalli (1996)	$O(n\alpha(n))$	$O(n^2\alpha(n))$	
Kaplan et al. (1999)		$O(n^2)$	
Bader et al. (2001)	$O(n)$		
Bergeron (2005)		$O(n^3)$	greatly simplified the result
Bergeron and Mixtacki (2004)	$O(n)$		further simplifications
Kaplan and Verbin (2005)		$O(n\sqrt{n \log n})$	almost always (unproven, empirical)
Tannier et al. (2007)		$O(n\sqrt{n \log n})$	
Han (2006)		$O(n\sqrt{n})$	
Swenson et al. (2010)		$O(n \log n)$	for most permutations (empirical)

2.3 Reversal Distance

The reversal model was introduced by Sankoff (1992). Three years later Hannenhalli and Pevzner (1995) devised a polynomial algorithm solving the sorting problem. This result was quite surprising; the algorithm was slow and the proof was difficult, but further results simplifying and speeding up the algorithm followed (see Table 2.1).

Today, we can compute the reversal distance in linear time and produce an optimal sorting sequence in $O(n^{1.5})$ time. Furthermore, empirical data suggest that in fact, we can sort most permutations in $O(n \log n)$ time. It remains an open problem to devise a deterministic algorithm that sorts all permutations in $O(n \log n)$ time.

We will assume that $\pi = (\overrightarrow{0}, \pi_1, \ldots, \pi_n, \overrightarrow{n+1})$ is a linear extension of a signed permutation. Reversal operation inverting the segment from i to j is a signed permutation

$$\rho(i,j) = (0, 1, \ldots, i-1, -j, -(j-1), \ldots, -i, j+1, \ldots, n+1),$$

so that

$$\pi \circ \rho(i,j) = (\overrightarrow{0}, \pi_1, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n, \overrightarrow{n+1}).$$

The reversal distance between π and γ, denoted $rev(\pi, \gamma)$, is the least number of reversals needed to transform one permutation into another; however, since $rev(\pi, \gamma) = rev(\gamma^{-1} \circ \pi, \iota)$, we will assume that the second permutation is the identity and we will write simply $rev(\pi)$ instead of $rev(\pi, \iota)$.

The reversal model and DCJ. We can also treat the reversal model as a restriction of the DCJ model. In particular, we only allow operations that do not create new chromosomes. Since the DCJ model supports reversals, but also various other operations, $dcj(\pi, \gamma) \leq rev(\pi, \gamma)$ and we immediately have a lower bound on the reversal distance.

This lower bound is not always tight. For example, in the DCJ model, we need 2 operations (one block interchange) to sort permutation $h = (\overrightarrow{0}, \overrightarrow{2}, \overrightarrow{1}, \overrightarrow{3})$, but we need 3 reversals. This is because in the DCJ model, we can create a new common adjacency in every step. On the other hand, in permutation h, it is not possible to heal any breakpoint by a single reversal.

Opposite and aligned pairs. This leads us to the question: *When is it possible to create a new common adjacency by reversal?* For two consecutive markers m and $m+1$, the answer is easy: we can create adjacency $m, m+1$ by a single reversal if and only if they have opposite orientation in π; all the four possible cases are:

$$(\overrightarrow{0},\ldots,\overrightarrow{m},\underline{\ldots\ldots,\overleftarrow{m+1}},\ldots,\overrightarrow{n+1}), \qquad (\overrightarrow{0},\ldots,\overleftarrow{m},\underline{\ldots\ldots,\overrightarrow{m+1}},\ldots,\overrightarrow{n+1}),$$

$$(\overrightarrow{0},\ldots,\underline{\overrightarrow{m+1},\ldots\ldots},\overleftarrow{m},\ldots,\overrightarrow{n+1}), \qquad (\overrightarrow{0},\ldots,\overleftarrow{m+1},\underline{\ldots\ldots,\overrightarrow{m}},\ldots,\overrightarrow{n+1}).$$

In each case, the segment that should be reversed is underlined; in the first row, we create adjacency $(\overrightarrow{m}, \overrightarrow{m+1})$, in the second row, we create adjacency $(\overleftarrow{m+1}, \overleftarrow{m})$.

If m and $m+1$ have opposite orientation in π, we call $(m, m+1)$ an *opposite pair*[2], otherwise $(m, m+1)$ is an *aligned pair*[3]. We can see that permutation h contains no opposite pairs and hence its reversal distance is bigger than the DCJ distance. However, even when π does contain opposite pairs, we have to be careful with the sorting. For instance, take $\pi = (\overrightarrow{0}, \overleftarrow{1}, \overleftarrow{3}, \overrightarrow{2}, \overrightarrow{4})$; the DCJ distance from the identity is 3 and it can be sorted by 3 reversals:

$$(\overrightarrow{0}, \underline{\overleftarrow{1}, \overleftarrow{3}}, \overrightarrow{2}, \overrightarrow{4}) \quad \rightsquigarrow \quad (\overrightarrow{0}, \overrightarrow{1}, \underline{\overleftarrow{3}, \overrightarrow{2}}, \overrightarrow{4}) \quad \rightsquigarrow \quad (\overrightarrow{0}, \overrightarrow{1}, \underline{\overleftarrow{2}}, \overrightarrow{3}, \overrightarrow{4}) \quad \rightsquigarrow \quad \iota$$

However, if we started differently and reversed $\overleftarrow{1}, \overleftarrow{3}$ in order to move $\overleftarrow{1}$ next to $\overrightarrow{2}$, we would get

$$(\overrightarrow{0}, \underline{\overleftarrow{1}, \overleftarrow{3}}, \overrightarrow{2}, \overrightarrow{4}) \quad \rightsquigarrow \quad (\overrightarrow{0}, \overrightarrow{3}, \overrightarrow{1}, \overrightarrow{2}, \overrightarrow{4}),$$

which is bad (there are no opposite pairs in the permutation and we actually need 3 more reversals to sort it). Note that both opposite pairs $(\overrightarrow{0}, \overleftarrow{1})$ and $(\overrightarrow{2}, \overleftarrow{3})$ changed into aligned pairs $(\overrightarrow{0}, \overrightarrow{1})$ and $(\overrightarrow{2}, \overrightarrow{3})$. The moral of this story is: 1. Opposite pairs are good. 2. Reversals may have side effects – they may turn an opposite pair into an aligned pair and vice versa. 3. We have to choose reversals carefully, otherwise we may get stuck with a permutation with all pairs aligned.

Breakpoint graph. In the reversals theory, it is customary to use breakpoint graphs instead of adjacency graphs, so let us make a little detour and define the breakpoint graphs before we study the effects of reversals.

In the *breakpoint graph* $BG(\pi)$, vertices are extremities of π (from the sentinels 0 and $n+1$ we only include 0^+ and $(n+1)^-$) and edges are of two types: *black* or *reality* edges are adjacencies in π and *grey* or *desire* edges are adjacencies in the identity permutation, i.e., they connect pairs of consecutive numbers (see Fig. 2.7 for an example of breakpoint graph).

[2] called *oriented pair* in literature

[3] called *unoriented pair* in literature; however, we find this terminology confusing

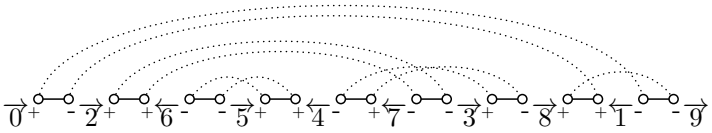


Figure 2.7: Breakpoint graph of $\pi = (\overrightarrow{0}, \overrightarrow{2}, \overleftarrow{6}, \overrightarrow{5}, \overleftarrow{4}, \overleftarrow{7}, \overrightarrow{3}, \overrightarrow{8}, \overleftarrow{1}, \overrightarrow{9})$; solid edges are the adjacencies of π (black edges), while dotted edges are the adjacencies of the identity (grey edges).

Breakpoint graphs are closely related to adjacency graphs, in fact, adjacency graph is a line graph of a breakpoint graph – for each edge in $BG(\pi)$ there is a vertex in $AG(\pi, \imath)$ and two vertices in $AG(\pi, \imath)$ are connected, if the corresponding edges are incident in $BG(\pi)$. Our result $rev(\pi) \geq dcj(\pi, \imath)$ can be translated into the language of a breakpoint graphs as:

$$rev(\pi) \geq n + 1 - c, \tag{$*$}$$

where c is the number of cycles in $BG(\pi)$.

Also note that if grey edge corresponds to an opposite pair, the incident black edges should be cut in order to move the pair together. We will denote v_i the grey edge connecting i^+ and $(i+1)^-$ and for opposite pair $(i, i+1)$, we will denote $\rho(v_i)$ the corresponding reversal that brings i and $i+1$ together.

Side effects of reversals. Notice that if two grey edges v_i and v_j are disjoint or nested, they do not affect each other. On the other hand, if they *overlap*, then v_i contains only one of the endpoints j and $j+1$. Thus, performing the reversal $\rho(v_i)$ changes the orientation of exactly one of the markers j and $j+1$ and the pair turns from opposite to aligned or from aligned to opposite. The overlap relation between grey edges is therefore important.

Moreover, if v_i overlaps edges v_j and v_k and we perform the reversal $\rho(v_i)$, the overlap relation between v_j and v_k changes – if v_j and v_k overlapped, they will not overlap and if they did not overlap, they will (see Fig. 2.8).

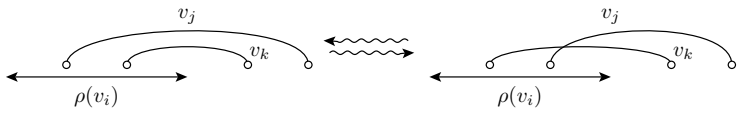
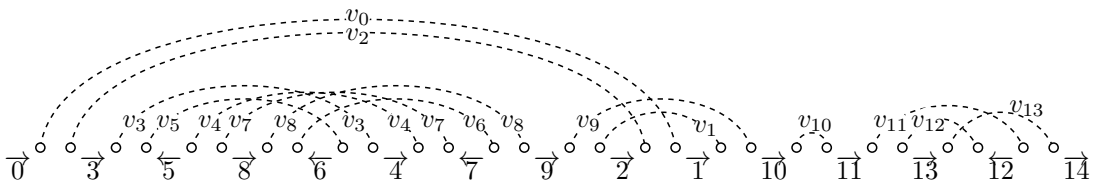


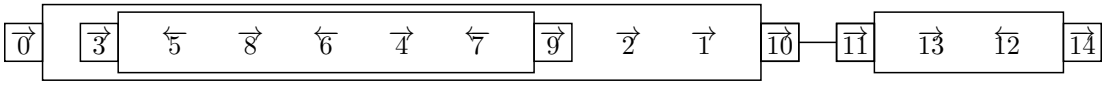
Figure 2.8: Effect of reversal on overlapping edges: Grey edge v_i overlaps edges v_j and v_k. If v_j and v_k do not overlap (left), they will overlap after the application of reversal $\rho(v_i)$ (right) and vice versa.

This motivates the introduction of an overlap graph – a structure that will capture all the grey edges – opposite and aligned pairs – and the overlapping relation between them.

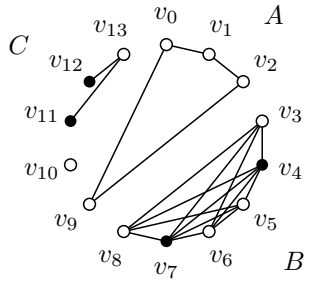
Overlap graph. The *overlap graph* of a permutation π is the graph $OV(\pi)$ whose vertices are the $n+1$ grey edges of $BG(\pi)$ and two vertices are connected, if the corresponding grey edges overlap. Furthermore, we will colour vertices corresponding to opposite pairs black and vertices corresponding



(a) Permutation π and the grey edges of its breakpoint graph.



(b) Framed common intervals of π depicted as boxes; the smaller boxes are frames of the FCIs. The smallest FCI is an adjacency, which consists of the frames only, here indicated by a line connecting $\overrightarrow{10}$ and $\overrightarrow{11}$.



(c) Overlap graph $OV(\pi)$ consisting of four connected components; v_{10} is an isolated vertex corresponding to adjacency $(\overrightarrow{10}, \overrightarrow{11})$, components B and C are good components, and A is a bad component (all its vertices are white).

Figure 2.9: Permutation π, its framed common intervals and its overlap graph.

to aligned pairs white. Note that an isolated vertex in $OV(\pi)$ corresponds to an adjacency in π. Thus, when we are sorting π into the identity permutation, we are trying to isolate all vertices.

Let v be a black vertex in $OV(\pi)$ representing an opposite pair in π. In the light of the above observations, we can think of performing the reversal $\rho(v)$ as follows: We are given a bicoloured graph G. When we "click" on a black vertex v, it becomes a white isolated vertex, the colour of all its neighbours is flipped and the set of edges between them is complemented. This operation is called *local complementation* of v in G and we denote the resulting graph by G/v.

The observations made above are formulated in the following lemma:

Lemma 3. *Let π be a signed permutation and let v be a vertex in $OV(\pi)$ corresponding to an opposite pair. Then performing the reversal $\rho(v)$ corresponds to local complementation of v's neighbourhood:*

$$OV(\pi \circ \rho(v)) = OV(\pi)/v.$$

The overlap graph may consist of several connected components (Fig. 2.9(c)). We say that a component is *good*, if it contains a black vertex, otherwise it is a *bad component* (unless it is just a single isolated vertex). When sorting a good component, we want to isolate black vertices by clicking on them, but at the same time, we try to turn other vertices black so that we do not get stuck with all vertices white.

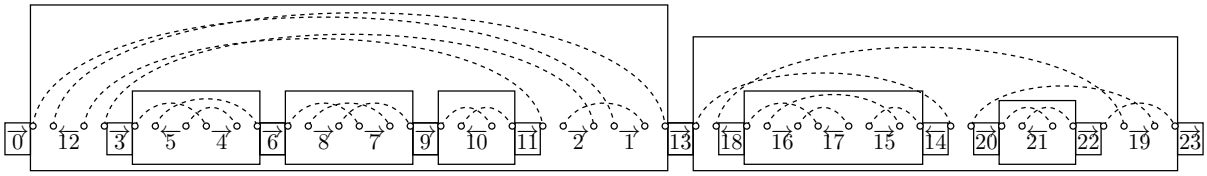


Figure 2.10: Framed common intervals of a permutation. Note that grey edges of one component are inside of one FCI and not inside the nested FCIs. Also notice the structure of FCIs: every two FCIs are either disjoint, nested, or linked. Interval from $\overrightarrow{3}$ to $\overrightarrow{11}$ is a chain of three linked FCIs; another chain consists of FCI $(\overrightarrow{0}, \overrightarrow{13})$ and $(\overrightarrow{13}, \overrightarrow{23})$.

We will describe how to do that properly in the next section and in the following section, we will describe how to cope with bad components, culminating in a formula for the reversal distance. However first, let us introduce the concept of framed intervals, that establishes a nice alternative characterization of good and bad components (and the other concepts we will run into).

Framed common intervals. An *interval* (or a *segment*) of a permutation π is a set $\{\pi_i, \pi_{i+1}, \ldots, \pi_{i+k}\}$. A set I is a *common interval* of π and ι, if I is an interval in both π and in ι. That is, if the set $\{\pi_i, \ldots, \pi_{i+k}\}$ is a set of consecutive numbers $\{m, m+1, m+2 \ldots, m+k\}$. Moreover, interval I is a *framed common interval (FCI)* of π and ι, if in addition to being a common interval, it either starts with the smallest and ends with the highest marker positively oriented: $\pi_i = \overrightarrow{m}$, $\pi_{i+k} = \overrightarrow{m+k}$, or ends with the smallest and starts with the highest marker negatively oriented: $\pi_i = \overleftarrow{m+k}$, $\pi_{i+k} = \overleftarrow{m}$. Furthermore, we will require that $|I| \geq 2$ and I is not a union of two such intervals (an example of an FCI is on Fig. 2.9(b)). Markers π_i and π_{i+k} are called *frames* of the interval. The smallest possible FCI consists of just these two frame markers and it corresponds to a common adjacency with ι.

The connection with overlap graphs is shown on Fig. 2.10. Since an FCI consists of consecutive numbers $\{m, \ldots, m+k\}$ and the smallest and the highest markers frame the interval, there is no grey edge going from within the FCI outside and grey edges of one FCI cannot overlap with grey edges outside of the interval. Consequently, a single FCI corresponds to one or several connected components of the overlap graph.

On the other hand, if we take an FCI and throw out all the nested FCIs, we end up with a single component. More precisely, we will say that an FCI $\{m, \ldots, m+k\}$ contains all the extremities inside it, excluding the outer extremities m^- and $(m+k)^+$. We say that extremity p *belongs* to the smallest FCI that contains it. All the grey edges connecting extremities belonging to a single FCI form a single connected component in $OV(\pi)$.

The relation of belonging is well-defined, since it can be easily proved that FCIs cannot overlap by more than one frame element. Two FCIs can be disjoint, nested, or they may overlap at exactly one frame element and then we say the FCIs are *linked*. A maximal sequence of linked FCIs is called a *chain* (see Fig. 2.10).

If we assume that all the nested FCIs of a given FCI are already sorted, or equivalently, if we treat every nested chain as a single element (with orientation of its frame elements), then the corresponding component is bad if and only if all the elements of the permutation have the same orientation. For example component $\overrightarrow{6}, \overrightarrow{8}, \overrightarrow{7}, \overrightarrow{9}$ in Fig. 2.10 is bad.

All the FCIs, good and bad components can be found in linear time as shown by Bergeron et al. (2002).

2.3.1 Sorting Good Components

Good components are the ones, which contain a black vertex. A reversal is called *safe*, if it does not create new bad components. In this section, we will show that given an overlap graph with no bad components, there is always a safe reversal corresponding to an opposite pair.

We will say that a graph is *tight*, if all its components are good. The main result of this section is:

Theorem 2. *If an overlap graph of permutation π is tight, the lower bound $(*)$ is tight, that is,*

$$rev(\pi) = dcj(\pi, \iota) = n + 1 - c,$$

where c is the number of cycles in $BG(\pi)$.

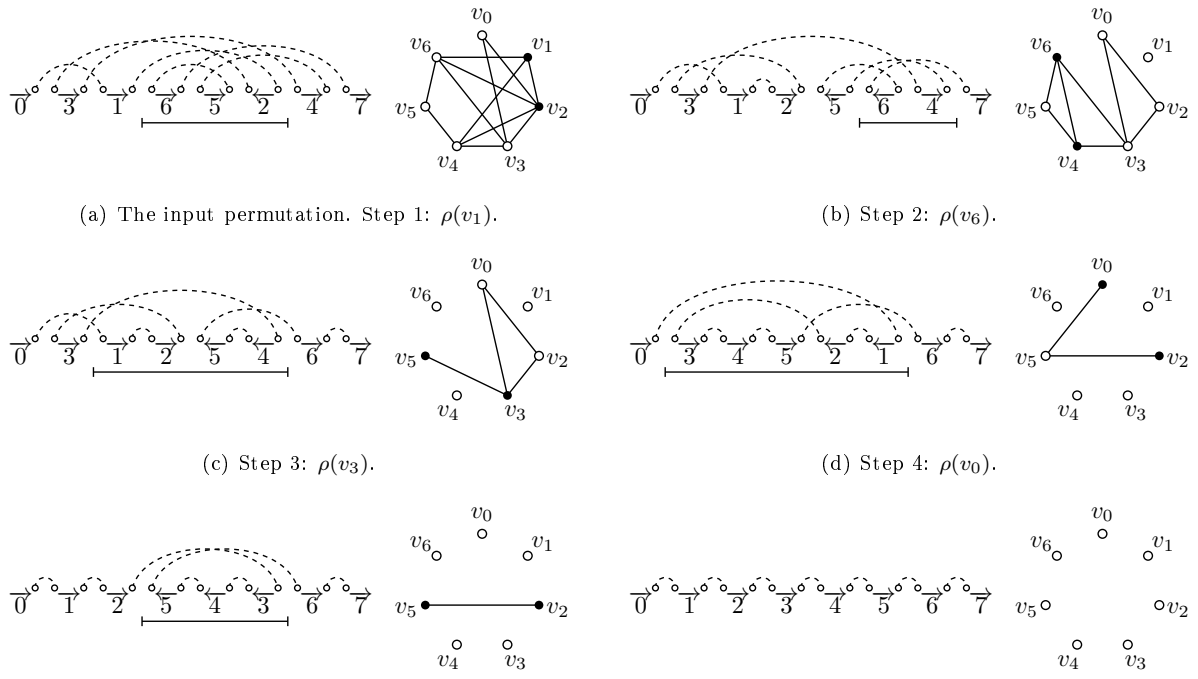
Local complementation game. Let us denote by $N[v]$ the *closed neighbourhood* of vertex v, i.e., the set of vertices adjacent to v with v itself. Local complementation of a black vertex v in G results in a new graph G/v, where the colours of vertices in $N[v]$ are flipped and the edges in $N[v]$ are complemented. We say that $s = u_1, \ldots, u_k$ is a *sequence of black vertices for* G, if for each i, u_i is a black vertex in $G/u_1/u_2/\cdots/u_{i-1}$.

We can formulate the problem of sorting good components as a *local complementation game*: Given a tight graph with n vertices, find a sequence of black vertices $u_1, u_2, \ldots, u_n$ for G such that $G/u_1/u_2/\cdots/u_n$ consists of n isolated vertices. If G is an overlap graph of some permutation, the sequence of reversals $\rho(u_1), \rho(u_2), \ldots, \rho(u_n)$ sorts the permutation. See Fig. 2.11 for an example of sorting a permutation by playing the local complementation game.

Approach of Bergeron (2005). Let v be a black vertex; its *score* is the number of black vertices in G/v. Bergeron (2005) proved that we can win the local complementation game by always choosing a black vertex with the maximum score. In other words,

Theorem 3 (Bergeron (2005)). *If u is a vertex in $OV(\pi)$ with the highest score, then reversal $\rho(u)$ is safe.*

Proof. Since $score(v) = t + w - b - 1$, where t is the total number of black vertices in G and b, w is the number of black and white neighbours of v, a vertex with the highest score is one with the highest difference between the number of white and black neighbours.



(a) The input permutation. Step 1: $\rho(v_1)$.

(b) Step 2: $\rho(v_6)$.

(c) Step 3: $\rho(v_3)$.

(d) Step 4: $\rho(v_0)$.

(e) Step 5: $\rho(v_2)$.

(f) Sorted permutation.

Figure 2.11: One way of sorting permutation $(\overrightarrow{0}, \overrightarrow{3}, \overrightarrow{1}, \overrightarrow{6}, \overrightarrow{5}, \overleftarrow{2}, \overrightarrow{4}, \overrightarrow{7})$ by 5 reversals.

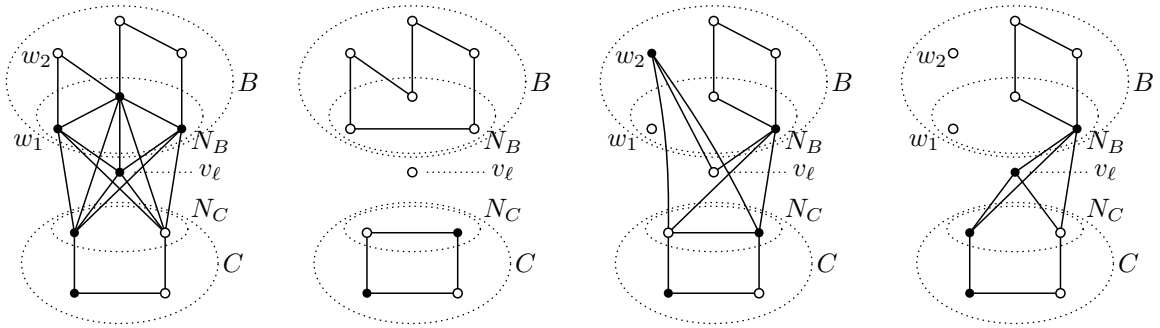
The proof is by contradiction: let u be a vertex with the highest score and C a bad component in G/u. Since G did not contain bad components, there must have been a black vertex v in C, which was adjacent to u in G. In fact, v must have been adjacent to all the white neighbours of u. Thus, if we denote b, w and b', w' the number of black and white neighbours of u and v respectively, we have $w' \geq w$.

Furthermore, all the black neighbours of v must have turned white, so they must have been neighbours of v and $b' \leq b$. From the two inequalities, we have $w' - b' \geq w - b$ and since u had the highest score, this is only possible if $w = w'$ and $b = b'$, which means that u and v share exactly the same neighbourhood. However, in this case, v becomes an isolated vertex in G/u, which is *not* a bad component – a contradiction. □

Approach of Tannier et al. (2007). The following approach, combined with an efficient data structure, constitutes the fastest approach to sorting good components so far. Let G be a tight graph and $u_1, \ldots, u_k$ a sequence of vertices; we will denote the subsequence $u_1, \ldots, u_i$ by s_i and we will write G/s_i for the graph $G/u_1/\cdots/u_i$. Let $s = u_1, \ldots, u_k$ be a sequence of black vertices for G, (i.e., u_i is black in G/s_{i-1}); we say that s is *maximal*, if we cannot extend it – there is no black vertex in G/s. We say that a sequence of black vertices is *total*, if it is maximal and G/s consists of isolated vertices only.

Theorem 4 (Tannier et al. (2007)). *Let G be a tight graph and s a maximal but not total sequence of black vertices. Then s can be split into two sequences $s = s^1, s^3$ and there is a nonempty sequence s^2 such that s^1, s^2, s^3 is a sequence of black vertices for G.*

Proof. Let B be the set of non-isolated vertices in G/s – these form the bad components we end up with – and let u_ℓ be the first vertex in sequence s such that all vertices of B are in bad components in G/s_ℓ. We will call G/s_ℓ a "bad" graph and denote it G_B (Fig. 2.12(b)). We will denote the previous graph, just before local complementation of v_ℓ, by G_1 (Fig. 2.12(a)). We split sequence s into $s^1 = u_1, \ldots, u_{\ell-1}$ and $s^3 = u_\ell, \ldots, u_k$ (thus $G_1 = G/s^1$ and $G_B = G_1/v_\ell$). Let C be the set of vertices in good components of G_1 – these are the vertices which become isolated by sequence s^3 – and finally, let N_B and N_C be the neighbours of v_ℓ in B and C, respectively (see Fig. 2.12(a)).



(a) G_1 (b) G_B (c) G_1/w_1 (d) $G_1/w_1/w_2$

Figure 2.12: Figure (a) shows graph G_1 that we obtain after application of s_1; (b) after local complementation of v_ℓ, we end up with a "bad" graph G_B; C and B consist of bad and good components, respectively; N_B and N_C are neighbours of v_ℓ in B and C in G_1. Figures (c) and (d) show that if we apply vertices w_1 and w_2, the subgraph induced by $C \cup \{v_\ell\}$ remains the same as in G_1 and thus we can continue with sequence s^3.

Since B consists only of bad components in G_B, the vertices of N_B must have been all black and vertices of $B - N_B$ must have been all white in G_1. Since B and C are disconnected in G_B, there must have been all possible edges between N_B and N_C in G_1. Finally, there must have been no edges going from $B - N_B$ outside of B and from $C - N_C$ outside of C.

In N_B, there must be a vertex w_1 such that w_1 and v_ℓ have different closed neighbourhoods, $N[w_1] \neq N[v_\ell]$ (otherwise N_B would be a clique isolated from $N - N_B$ and after the local complementation of v_ℓ, N_B would be a set of isolated vertices, which is a contradiction). Moreover, there must be a vertex w_2 in the symmetric difference $N[w_1] \oplus N[v_\ell]$. It can be easily proved that after local complementation of w_1 and w_2 the subgraph induced by $C \cup \{v_\ell\}$ remains unchanged (compare Fig. 2.12(a) and Fig. 2.12(d)), so s^3 is a sequence of black vertices in $G/w_1/w_2$ and we may extend sequence s by inserting the two-element sequence $s^2 = w_1, w_2$. □

Tannier et al. (2007) use a data structure by Kaplan and Verbin (2005) or Han (2006) to maintain information about all vertices. They can choose and perform a single reversal in $O(\sqrt{n})$ time and repair the sequence in $O(\sqrt{n})$. Swenson et al. (2010) only maintain information about the first opposite pair

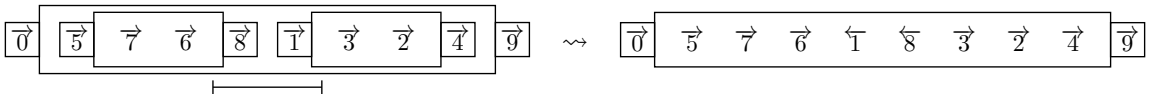
so they can perform a single reversal in $O(\log n)$. When they get stuck with all-white components, the recovery is costly – in linear time. However, their experiments with sorting random permutations suggest that the number of repairs is usually very small (constant average and variance).

2.3.2 Sorting Bad Components

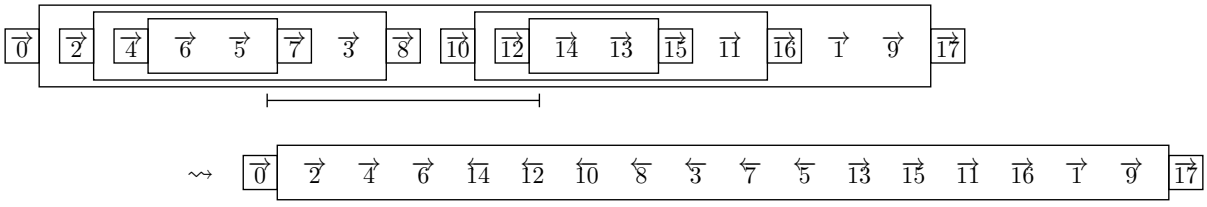
Bad components are FCIs where all elements have the same orientation (nested chains are treated as single elements). In this section, we will describe how to get rid of bad components. In fact, this can be done at the beginning in linear time so the hard part is actually sorting the good components (without creating new bad ones).

Cutting and merging bad components. If B is a bad component, then any reversal acting on two black edges belonging to one cycle in B turns it to a good component and leaves the number of cycles of the permutation unchanged. In particular, the reversal corresponding to an aligned pair (π_i, π_{i+1}) in a bad component is such a reversal. This actually corresponds to "clicking" on a white vertex w in the overlap graph of π, which results in complementation of w's neighbourhood and flipping w's colour. (Unlike clicking on a black vertex, here, we do not remove edges incident to w; so when we click on a white vertex, we can turn a bad component into a good one, but this move does not isolate the vertex.) This operation is called *cutting* the bad component.

The other operation, called *merging*, acts on two black edges from two different bad components. In particular, we can take a reversal ranging from one frame to another as in the following example:



Note that merging two components actually merges two cycles in the breakpoint graph (which is bad – when sorting, we try to increase the number of cycles). So in this case, cutting both components with two reversals would have the same effect. However, the advantage of merging is that by one operation, we actually destroy all the bad components which have one frame in the interval:



Component tree. Let us represent the components with the nesting relation by a *component tree* T_π, which is defined as follows:

- each component is represented by a *round* node; good components and adjacencies are black and bad components are white;
- each chain of components is represented by a *square* node whose children are round nodes representing the linked components (in that order);
- each square node is a child of the smallest component that contains the chain.

Let A and B be two components, vertices in T_π, and notice the unique path connecting A and B. If the highest vertex on the path is round, it corresponds to the smallest common component C, which includes both A and B. Otherwise, if the highest vertex is square, A and B are nested in two components of a single chain and no component on the path contains both A and B.

As we noted above, there are two strategies for destroying bad components – cutting, which fixes the corresponding vertex in T_π and merging, which fixes the corresponding two vertices, but also as a bonus, it fixes all the bad components on the unique path connecting these two components.

Covers. This motivates the introduction of path covers: a *path cover* of the component tree T_π is a collection of paths connecting all the bad components of π, such that each terminal node of a path belongs to a unique path.

A path consisting of a single bad component, called a *short* path, corresponds to cutting the bad component and it is assigned cost 1. A path consisting of two or more bad components is a *long* path and corresponds to merging the terminal nodes. The cost of a long path is 2, since merging decreases the number of cycles.

Thus, each cover of T_π corresponds to a sequence of cutting and merging reversals that fix all the bad components in π. The *cost* of a cover is the sum of the costs of its paths; it is *optimal*, if it has minimal cost.

Theorem 5 (Bergeron and Mixtacki (2004)). *If a permutation π has c cycles in the breakpoint graph and an optimal path cover of its component tree T_π costs t,*

$$rev(\pi) = dcj(\pi, \iota) + t = n + 1 - c + t.$$

The reversal distance formula. The last problem is how to compute the optimal path cover for a given component tree. We state the result without proof.

Let T' be the smallest *unrooted*[4] subtree of T_π containing all the bad components of π (remove all the dangling black nodes and square nodes from T_π). Define a *branch* of a tree as the set of nodes from a leaf up to, but excluding, the next node of degree at least 3. A *short* branch of T' contains only a single bad component while a *long* branch contains two or more bad components.

[4]since we extend our permutation by $\overrightarrow{0}$ and $\overrightarrow{n+1}$, the whole permutation is an FCI, which constitutes a root of T_π; here we treat T_π as unrooted, so the root actually counts as one of the leaves

Theorem 6 (Bergeron and Mixtacki (2004)). *Let T' be the smallest unrooted subtree of T_π containing all the bad components; let T' have ℓ leaves.*

- *If ℓ is even or if there is a leaf on a short branch in T', $t = \ell$,*
- *otherwise, $t = \ell + 1$.*

2.4 Reversal-Translocation Distance

As we have mentioned in Section 1.2, nuclear genomes of eukaryotic organisms usually consist of multiple linear chromosomes. These genomes evolve mainly by reversals and translocations (and fusions and fissions, which are considered as a special case of translocation). The smallest number of reversals and translocations is the *reversal-translocation distance*, or simply *RT-distance*.

History. The first solution was given by Hannenhalli and Pevzner (1995) who also introduced the problem; hence the RT-model is also called the Hannenhalli-Pevzner, or HP-model; RT-distance is also called HP-distance. This result has a quite long history of "debugging" – first errors and gaps were found and fixed by Tesler (2002), who implemented the algorithm in a software called `GRIMM`. Another counter-example was found by Ozery-Flato and Shamir (2003) and the RT-distance formula was corrected again. This was the presumably correct formula until Jean and Nikolski (2007) found another bug and patched the result.

We will present the results of Bergeron et al. (2008) who tried hard to simplify the result using their insights from DCJ sorting, FCIs, and path covers. They extended the result on sorting by reversals and formulated the RT-distance formula as

$$rt(\pi, \gamma) = dcj(\pi, \gamma) + t,$$

where t is the cost of an optimal path cover of a certain component tree $T_{\pi,\gamma}$.

RT via DCJ. We will treat the RT-model as a restriction of the more general DCJ model to multilinear genomes. A DCJ operation is called *linear*, if it does not create circular chromosomes. Given a multilinear genome, the linear operations are exactly reversals and translocations. The RT-distance between two multilinear genomes π and γ is the minimum number of linear DCJ operations needed to transform π into γ; we immediately have that $dcj(\pi, \gamma) \leq rt(\pi, \gamma)$.

FCIs and components. An *interval* $I = \{\ell, \ldots, r\}$ in π is a set of consecutive markers or telomeres within a chromosome of π. Telomeres and extremities of these markers form a path in π and the endpoints of this path are called *frames* of the interval. A *framed common interval (FCI)* of π and γ is a set $\{\ell, \ldots, r\}$ that is an interval with the same frames in π and γ and is not the union of two such intervals. Two FCIs are either disjoint, nested, or overlap at exactly one marker.

We say that an adjacency *belongs* to the smallest FCI that contains it and all the adjacencies that belong to a single FCI constitute a *component*. The adjacency graph $AG(C)$ of a component C is the subgraph of the adjacency graph $AG(\pi, \gamma)$ induced by the adjacencies of C. It can be proved that $AG(C)$ consists of a union of (whole) paths and cycles of $AG(\pi, \gamma)$.

We say that a component C is good, if there is a linear DCJ operation increasing the number of cycles in $AG(C)$. It can be shown that C is good if and only if

- it contains two elements with different orientation (nested chains are treated as single elements),
- or if $AG(C)$ contains two paths of even length.

Otherwise, the component is bad. The lower bound given by the DCJ distance is tight,

$$rt(\pi, \gamma) = dcj(\pi, \gamma),$$

if and only if there are no bad components.

Component tree. Given a chromosome C of genome π and its components relative to genome γ, define the forest $F_{C,\gamma}$ by the following construction:

- each component is represented by a round node; good components and adjacencies are black, bad components without telomeres are white, and bad components with one or two telomeres are grey;
- every chain is represented by a square node whose children are the round nodes representing the linked components (in that order);
- each square node is the child of the smallest component that contains the chain.

Suppose that genome π consists of chromosomes $\{C_1, C_2, \ldots, C_k\}$. The tree $T_{\pi,\gamma}$ is given by the following construction:

- the root is a black round node;
- all trees in the forests $F_{C_1,\gamma}, F_{C_2,\gamma}, \ldots, F_{C_k,\gamma}$ are children of the root.

Cutting and merging of components. If C is a bad component, then any reversal acting on adjacencies of the same cycle or the same path in $AG(C)$ turns C into a good component, but leaves the number of paths and cycles in $AG(C)$ unchanged. Such an operation is called *cutting* the bad component.

A DCJ operation acting on adjacencies of two different bad components A and B destroys or fixes all the components on the path from A to B in $T_{\pi,\gamma}$ without creating any new bad components. This operation is called *merging* of the bad components.

Covers. Let T' be the smallest subtree of $T_{\pi,\gamma}$ that contains all the bad components, that is, all the white and grey nodes. A *path cover* of T' is a collection of paths connecting all the bad components, such that each terminal node of a path belongs to a unique path.

A path consisting of just a single vertex is a *short* path and a path with two grey endpoints is a *grey* path. All the other paths are *long*. The cost of short and grey paths is 1, while the cost of long paths is 2. The cost of a cover is the sum of the costs of its paths and a cover is optimal, if it has minimal cost.

Theorem 7 (Bergeron et al. (2008)). *If t is the cost of an optimal cover of T', the smallest subtree of $T_{\pi,\gamma}$ that contains all the bad components, then:*

$$rt(\pi, \gamma) = dcj(\pi, \gamma) + t.$$

An optimal cover can be found in linear time (Erdös et al., 2011).

2.5 Other Distances

In the final section of this chapter, we very briefly review other rearrangement models. For a more comprehensive survey of the field, see the excellent book by Fertin et al. (2009), *Combinatorics of Genome Rearrangements*.

Distances between unsigned permutations. If the orientation of markers is not known (this may be the case for instance, if the data come from *in situ hybridization*), we can model genomes by classical (unsigned) permutations.

Unlike sorting signed permutations, sorting by reversals is NP-hard for unsigned permutations as shown by Caprara (1997). For signed permutations, we gave a lower bound $rev(\pi) \geq n + 1 - c$, where c is the number of cycles in the breakpoint graph $BG(\pi)$. An analogous result holds for unsigned permutations.

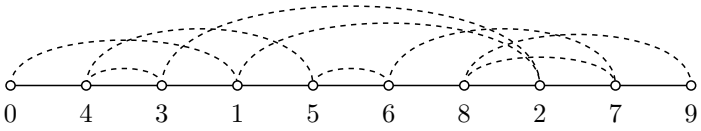
Recall that when π is a signed permutation, every element π_i corresponds to two vertices, π_i^- and π_i^+, so each vertex in the $BG(\pi)$ is incident with one black and one grey edge. Consequently, $BG(\pi)$ consists of alternating (black-and-grey) cycles only. However, when π is unsigned, each element corresponds to a single vertex of degree 4 in $BG(\pi)$ and there are many different decompositions of $BG(\pi)$ into a set of alternating edge-disjoint cycles (see Fig. 2.13). The problem of decomposing a given breakpoint graph into maximum number of alternating cycles is NP-hard.

The complexity of sorting by transpositions was a long standing open problem since Bafna and Pevzner (1998) introduced it. A recent paper by Bulteau et al. (2012b) proves that sorting by transpositions is NP-hard.

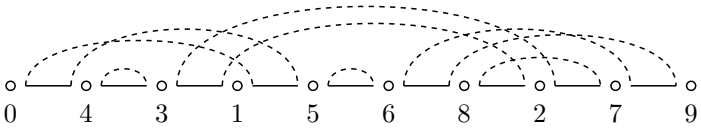
By contrast, sorting by block interchanges is easy: Let us treat permutation π as a *signed* permutation, add the sentinels, and compute its DCJ distance from identity. The DCJ distance on linear genomes is the minimum weight of a sequence of reversals (weight 1) and block interchanges (weight 2). However,

Table 2.2: Distances between unsigned permutations.

Model	Distance	Sorting	Notes, References
breakpoint	$O(n)$	—	Sankoff and Blanchette (1997) see Section 2.1
reversal		NP-hard	Caprara (1997); not approximable within 1.0008, thus APX-hard (Berman and Karpinski, 1999); $^{11}/_{8}$-approximable (Berman et al., 2002)
transposition		NP-hard	Bulteau et al. (2012b); $^{11}/_{8}$-approximable (Elias and Hartman, 2006)
block interchange	$O(n)$	$O(n \log n)$	Christie (1996); Feng and Zhu (2007)



(a) Breakpoint graph of the unsigned permutation π; compare this with breakpoint graph of a signed permutation on Fig. 2.7 with unique alternating cycle decomposition.



(b) The same breakpoint graph redrawn to display its decomposition into the maximum number of edge-disjoint alternating cycles. For unsigned reversal distance, we have $rev(\pi) \geq n + 1 - c^*$; in this case $c^* = 5$ and $rev(\pi) = 4$.

Figure 2.13: Unsigned permutation $\pi = (4, 3, 1, 5, 6, 8, 2, 7)$ and its breakpoint graph.

as we found out in Section 2.3, no reversal is optimal (in the DCJ model), if all the elements have the same orientation. Thus, the block interchange distance is exactly the DCJ distance divided by 2.

Distances between signed permutations. The breakpoint distance and conserved interval distance are distance measures between signed permutations with no underlying rearrangement operations. Conserved interval distance was introduced by Bergeron et al. (2002) and is computed as $ci(\pi, \gamma) = f(\pi) + f(\gamma) - 2f(\gamma^{-1} \circ \pi)$, where $f(\pi)$ is the number of FCIs in π.

We have studied the reversal distance in Section 2.3. Even though the distance and one sorting scenario can be computed in polynomial time, there are usually exponentially many different optimal sorting scenarios. Some of the scenarios may be more preferable (from the biological point of view) and therefore, it makes sense to add some restrictions to the model.

Table 2.3: Distances between signed permutations.

Model	Distance	Sorting	Notes, References
breakpoint	$O(n)$	—	Sankoff and Blanchette (1997) see Section 2.1
conserved intervals	$O(n)$	—	Bergeron et al. (2002)
reversal	$O(n)$	$O(n^{1.5})$	Hannenhalli and Pevzner (1999); Bader et al. (2001); Tannier et al. (2007); see Section 2.3
reversal--transposition	open		there are several variants; $(1+\epsilon)$-approximable, if a transposition has weight 2 (Eriksen, 2002)
reversals and block interchanges	$O(n)$	$O(n \log n)$	if block interchanges have weight 2 see Chapter 6
S-perfect reversals	NP-hard		even if S is nested (Figeac and Varré, 2004); polynomial if S is separable (Bérard et al., 2007)

One such restriction is called *perfectness*. If I is a common interval and ρ is a reversal, such that the reversed interval and I overlap, we say that ρ *breaks* the common interval I. A perfect scenario is one that does not break common intervals. More generally, if S is some subset of common intervals, a scenario is *S-perfect*, if the reversals do not break any interval in S.

Figeac and Varré (2004) proved that computing the S-perfect reversal distance is NP-hard, even if S is *nested*, i.e., no pair of intervals in S overlap. On the other hand, Bérard et al. (2007) showed that the distance can be computed in polynomial time if S is *separable*. A common interval is *strong*, if it does not overlap any other common interval and we say that S is separable, if every strong interval of S is the union of two overlapping intervals from S.

Different models arise when we combine reversals with another rearrangement operation. If we add block interchanges of weight 2 to reversals, we get the DCJ distance. Reversals combined with transpositions are more difficult. Several approximation results are known for different variants – depending on the weight of transpositions and whether a *transreversal* operation is allowed (transposition followed by reversal of the transposed segment as one operation).

Multichromosomal models. Multichromosomal genomes are modeled as collections of paths or cycles. We have studied the RT-distance in the previous section. A model containing only translocations alone was also studied. For unsigned genomes, the problem is NP-hard and not approximable within

Table 2.4: Multichromosomal models.

Model	Distance	Sorting	Notes, References
breakpoint	$O(n)$	—	Tannier et al. (2009), see Section 2.1
translocations	$O(n)$	$O(n^{1.5})$	Bergeron et al. (2006a), Ozery-Flato and Shamir (2006)
reversals--translocations	$O(n)$	$O(n^{1.5})$	Hannenhalli and Pevzner (1995), Bergeron et al. (2009), see Section 2.4
DCJ	$O(n)$		Bergeron et al. (2006b), see Section 2.2
restricted DCJ	$O(n)$	$O(n \log n)$	see Chapter 6
S-perfect DCJ	NP-hard		for weakly separable S, but polynomial for nested S (Bérard et al., 2009)
k-break	$O(n^{k-2}+n)$		Alekseyev and Pevzner (2008)

$^{5717}/_{5716}$ (Zhu and Wang, 2006) but $(1.5+\varepsilon)$-approximable (Cui et al., 2008), while for signed genomes, it is solvable in polynomial time (Bergeron et al., 2006a). A variation of the problem is sorting by *translocations preserving centromeres*. A centromere is a region of DNA involved in cell division (we can model it by a special vertex); we say that a genome is *legal*, if every chromosome contains exactly one centromere and a translocation preserves centromeres, if the resulting genome is legal. Ozery-Flato and Shamir (2007) devised a polynomial-time algorithm for sorting by translocations preserving centromeres.

We have described the DCJ model in Section 2.2. An interesting result on *perfect* DCJ distance was shown by Bérard et al. (2009): unlike perfect reversal distance, computing S-perfect DCJ distance is polynomial for nested S and NP-hard for weakly separable S (S is weakly separable, if every strong interval of length *at least 3* in S is the union of two overlapping intervals from S).

The DCJ model was further generalized by Alekseyev and Pevzner (2008) to a k-break model, which breaks the genome at k different places and joins the $2k$ extremities in arbitrary way. They give linear-time algorithms for computing the distance for any fixed k and a dynamic programming solution running in time $O(n^{k-2}+n)$.

Distances between arbitrary strings. As we mentioned in Section 1.2, genomes evolve not only by rearrangement operations, but also by duplications, insertions, deletions, or replacements. In this case, we may model unichromosomal genomes as strings, or signed strings over an alphabet of genes.

There are basically two approaches to defining a distance between arbitrary strings. In the *block edit models*, we add new operations such as duplications or deletions to the panoply of rearrangement operations and we define the distance as the minimum number of operations required to transform one genome into another. In the *match-and-prune models*, we first try to find the best matching of the

corresponding markers in both genomes (for instance, if π and γ contain two copies of marker m, we first try to decide, which copy in π corresponds to which copy in γ), then we remove all the unmatched markers and compute a classical rearrangement distance between the matched-and-pruned genomes.

Unfortunately, computing distance in most of these models is NP-hard and even inapproximable. To give a taste of what kinds of problems are studied in this field, we describe tree examples. Many more results can be found in the monograph by Fertin et al. (2009).

In the *exemplar breakpoint model*, we are given two strings s and t and the task is to remove all but one copy of each symbol in s and t in such a way that the breakpoint distance of the resulting permutations is minimized. This problem is NP-hard (Bryant, 2000) and even not approximable at all (Chen et al., 2006).

Our second example is the *minimum common string partition problem (MCSP)*: Two strings are *balanced* when they have the same number of occurrences of each symbol; we denote the number of occurrences of the most frequent symbol by c. A *partition* of a string s is a set of substrings $\{s_1, \ldots, s_p\}$ such that the concatenation of $s_1 \cdots s_p$ is s.

In the MCSP problem, we are given two balanced strings s and t and we seek to find the smallest set of substrings that is a partition of both s and t. This problem is APX-hard even if $c = 2$, i.e., every symbol occurs at most twice in s and t (Goldstein et al., 2005). The problem is also NP-hard when there is only one symbol occurring more than once (Blin et al., 2004). On the other hand, the problem is 1.1037-approximable for $c = 2$, 4-approximable for $c = 3$ (Goldstein et al., 2005), and both $4c$-approximable and $O(\log n \log^* n)$-approximable in general (Kolman and Walen, 2007).

Our third example is sorting signed strings by reversals. This problem is NP-hard even for binary alphabets and it is also NP-hard if there is at most one positive and one negative occurrence of each symbol (Radcliffe et al., 2006). The problem is $O(n^{0.69})$-approximable (Chrobak et al., 2005) and $O(c)$-approximable.

Chapter 3

Median Problem

Median. In this chapter, we will study the median problem for different rearrangement distances. Recall that in the median problem, we are given genomes π_1, π_2, and an outgroup genome π_3, and we try to reconstruct the genome of the common ancestor of π_1 and π_2. The most parsimonious solution, a genome that minimizes the total distance from π_1, π_2, and π_3, is called *median*.

Throughout the chapter, π_1, π_2, and π_3 will denote the input genomes. We define the *score* of M as

$$S(M) = d(M, \pi_1) + d(M, \pi_2) + d(M, \pi_3),$$

so that median is a genome with the minimum score. We denote this score and the set of all medians by

$$S^* = S^*(\pi_1, \pi_2, \pi_3) = \min_M S(M) \qquad \text{and} \qquad \mathfrak{M} = \mathfrak{M}(\pi_1, \pi_2, \pi_3) = \{\, M \mid S(M) = S^* \,\}.$$

Apart from reconstructing the ancestral genomes, medians were also used to model consensus genomes when different sources are inconsistent (Jackson et al., 2007). Bourque et al. (2005) tried to infer statistics on the rearrangement rates on different lineages using medians. Finally, median solvers are often used repeatedly when addressing more general problems such as small and large phylogeny, which we review in Chapter 5.

Bounds. As we shall see in the next few sections, computing median is usually a hard problem. However, our search may be guided by the following lower bound which easily follows from the triangle inequality:

$$S^* \geq \left\lceil \frac{d_{1,2} + d_{2,3} + d_{3,1}}{2} \right\rceil, \qquad \text{where } d_{i,j} = d(\pi_i, \pi_j). \tag{$*$}$$

A median is called *perfect*, if this lower bound is reached.

On the other hand,

$$S^* \leq \min\Big\{ S(\pi_1), S(\pi_2), S(\pi_3) \Big\} = \min\Big\{ (d_{1,2} + d_{1,3}), (d_{2,1} + d_{2,3}), (d_{3,1} + d_{3,2}) \Big\}.$$

From the lower bound it follows that a trivial algorithm, which picks one of the genomes π_1, π_2, or π_3 with the minimum score, is in fact a $4/3$-approximation algorithm.

Table 3.1: Complexity of computing median in various genome models. The unichromosomal models may be linear or circular – the problems are equivalent. The multichromosomal models may be circular or mixed.

Genome Model		Complexity	Notes, references
breakpoint	unichromosomal	NP-hard	(Pe'er and Shamir, 1998; Bryant, 1998), Section 3.1
	multilinear	NP-hard	(Tannier et al., 2009)
	multichromosomal	$O(n^3)$, $O(n\sqrt{n})$	(Tannier et al., 2009), Chapter 8
DCJ	unichromosomal	NP-hard	(Caprara, 2003), Section 3.2
	multilinear	NP-hard	(Tannier et al., 2009)
	multichromosomal	NP-hard	Chapter 6
RT	unichromosomal	NP-hard	(Caprara, 2003)
	multilinear	NP-hard?	open

Circular genomes and matchings. In this chapter, we consider mainly genomes with (possibly multiple) circular chromosomes, which are easier to work with. Recall that the set of adjacencies in such genomes forms a perfect matching of all extremities. (From now on, we will refer to perfect matchings simply as matchings.)

Note well the difference between unichromosomal and multichromosomal models – in the former, we require the median to be unichromosomal. For example in the unichromosomal DCJ model, given three circular genomes

$$\pi_1 = [\overrightarrow{1}, \overrightarrow{2}, \overrightarrow{3}, \overrightarrow{4}], \qquad \pi_2 = [\overrightarrow{2}, \overrightarrow{1}, \overrightarrow{3}, \overrightarrow{4}], \qquad \pi_3 = [\overrightarrow{2}, \overrightarrow{3}, \overrightarrow{1}, \overrightarrow{4}],$$

any one of the input genomes is also median with median score $0 + 2 + 2 = 4$ (we can get one genome from another by one block interchange). However, if multiple chromosomes are allowed, genome $M = [\overrightarrow{1}], [\overrightarrow{2}, \overrightarrow{3}, \overrightarrow{4}]$ has score $1 + 1 + 1 = 3$ (incorporating $[\overrightarrow{1}]$ at the proper place takes just one operation).

Outline of this chapter. In the next two sections, we study the median problem in the breakpoint, DCJ, and RT model defined in the previous chapter. Unfortunately, the median problem is NP-hard in almost all of the studied models, see Table 3.1. Thus, we will also focus on fast exact and heuristic solutions.

3.1 Breakpoint Median

Breakpoint median problem was introduced by Sankoff and Blanchette (1997) and Blanchette et al. (1997). Pe'er and Shamir (1998) and Bryant (1998) proved that the problem is NP-hard for unichromosomal models; Pe'er and Shamir (2000) and Caprara (2002) then studied some approximation algorithms, Sankoff and Blanchette (1997) and Bryant (2004) gave a better lower bound for breakpoint median.

3.1.1 Unichromosomal Breakpoint Median

Properties of breakpoint median. A signed breakpoint median always contains the edges that are shared by all three input genomes, i.e., $(\pi_1 \cap \pi_2 \cap \pi_3) \subseteq M$ for all $M \in \mathfrak{M}$. This holds in both linear and circular breakpoint model. However, for unsigned genomes, this does not hold so strictly; there is always a median containing all shared edges, but there may be other medians which do not.

On the other hand, the dual inclusion $M \subseteq (\pi_1 \cup \pi_2 \cup \pi_3)$ does not always hold. For example, if

$$\pi_1 = [1, 6, 7, 8, 5, 2, 3, 4, 9] \qquad \pi_2 = [1, 2, 6, 7, 5, 3, 4, 8, 9] \qquad \pi_3 = [1, 2, 3, 6, 5, 4, 7, 8, 9],$$

then $\mathfrak{M} = \{[1, 2, 3, 4, 5, 6, 7, 8, 9]\}$, even though $(4, 5)$ does not belong to any of π_1, π_2, π_3.

Since breakpoint distance is a distance measure, lower bound $(*)$ on the breakpoint median holds. A better lower bound was given by Sankoff and Blanchette (1997): Let λ_i denote the number of distinct adjacencies shared by i input genomes, let n be the number of markers and let $\Lambda = \Lambda(\pi_1, \pi_2, \pi_3) = 2n - 2\lambda_3 - \lambda_2$. Then $S^* \geq \Lambda$ and this bound is realized if and only if there is a genome $M \subseteq (\pi_1 \cup \pi_2 \cup \pi_3)$ that contains all the adjacencies shared by two or three input genomes.

Exact algorithms. Sankoff and Blanchette (1997) provide a simple reduction from the breakpoint median problem to the traveling salesman problem (TSP). Given three unsigned circular genomes π_1, π_2, π_3, let G be a complete graph whose vertices are markers and edges have weight $w(x, y) = 3 - m(x, y)$, where $m(x, y)$ is the number of input genomes, which share adjacency xy. Then finding an unsigned circular median is equivalent to finding a Hamiltonian cycle of minimum length.

In the signed model, G is a complete graph on extremities, edges are weighted in the same manner, we just put sufficiently small weights $w(m^-, m^+)$ on the marker edges to ensure that they are chosen in the Hamiltonian cycle.

Sankoff and Blanchette (1997) use a branch and bound algorithm based on the lower bound $S^* \geq \Lambda$. (The lower bound can be generalized to the case when some adjacencies of the median are already fixed.)

Complexity. The breakpoint median problem is NP-hard for signed (unichromosomal) circular genomes. The proof by Bryant (1998) is by reduction from Directed-Hamiltonian-Cycle problem which is still NP-hard for graphs with maximum degree 3 (Garey and Johnson, 1979). If G is such directed graph, we can construct a three-coloured multigraph G', which can be decomposed into three monochromatic Hamiltonian cycles (these correspond to the input genomes π_1, π_2, π_3). Furthermore, each Hamiltonian cycle of G will correspond to a Hamiltonian cycle of G', which contains one from each set of parallel edges. This in turn corresponds to a unique median of π_1, π_2, and π_3 with score $S^* = \Lambda$ (it contains all edges of weight 2 or 3 and no edges of weight 0). Thus, if we could efficiently find a breakpoint median or at least decide whether $S^* = \Lambda$, we could also decide the Directed-Hamiltonian-Cycle problem.

We construct graph G' by first substituting each vertex in G for a path of length 2 to get graph G_2. It is straightforward to colour its edges, possibly adding parallel edges so that edges of one colour form either whole or a subset of Hamiltonian cycle. We finish the Hamiltonian cycles by connecting the fragments

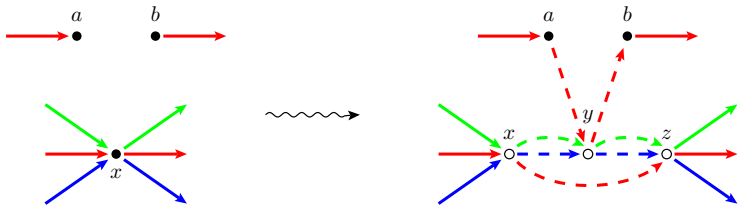


Figure 3.1: Suppose that one red fragment ends at a, another begins at b, and x is another vertex different from a and b (left). By splitting x into three vertices and adding the dashed edges (right) we connect these red fragments. This way we can gradually transform graph G_2 into G' consisting of three monochromatic Hamiltonian cycles. Notice that a median with score $S^* = \Lambda$ exists, it must contain edges $x \to y \to z$, which are shared by green and blue genomes (it must not contain edges $a \to y \to b$).

as shown in Fig. 3.1. Each monochromatic Hamiltonian cycle of the resulting graph G' corresponds to one input genome (more precisely, these are the monochromatic matchings after we replace each vertex x by a pair x^-, x^+ connected with a marker edge $x^- \to x^+$).

Thus, the median breakpoint problem for signed genomes is NP-hard. It remains NP-hard even with the constraint that $M \subseteq (\pi_1 \cup \pi_2 \cup \pi_3)$ and it is also NP-hard to decide given three genomes and their median whether the median is unique (consequently it is NP-hard to determine those edges which are shared by all medians).

Other results. Not surprisingly, the problem is also NP-hard for linear genomes. The median problems are essentially equivalent for signed and unsigned genomes: Let π be a signed circular genome and let $f(\pi) = \pi \cup B$. If we treat each extremity m^-, m^+ as a single marker, we get an unsigned genome with twice as many markers. It is not hard to prove that $S^*(\pi_1, \pi_2, \pi_3) = S^*(f(\pi_1), f(\pi_2), f(\pi_3))$ and $\mathfrak{M}(\pi_1, \pi_2, \pi_3) = \mathfrak{M}(f(\pi_1), f(\pi_2), f(\pi_3))$.

For signed permutations, the median problem is $7/6$-approximable (Pe'er and Shamir, 2000), for unsigned permutations, it is $5/3$-approximable (Caprara, 2002).

Bryant (2004) gave a better lower bound for breakpoint median: let $bp_m(\pi, \gamma) = 0$, if marker m is followed by the same marker in π as in γ, and let $bp_m(\pi, \gamma) = 1$ otherwise. Let π_1, π_2, and π_3 be the input permutations. For any permutation M define $bp_m(M) = bp_m(M, \pi_1) + bp_m(M, \pi_2) + bp_m(M, \pi_3)$; then

$$S^* = \min_M \sum_m bp_m(M) \geq \sum_m \min_M bp_m(M).$$

Bryant (2004) showed how to compute this lower bound in polynomial time and how to improve it using Lagrange multiplier techniques.

3.1.2 Multichromosomal Breakpoint Median

Tannier et al. (2009) proved that the median problem is still NP-hard for multilinear breakpoint model. However, for multichromosomal circular or mixed genomes, the problem is polynomially solvable. Note

that these are the only interesting genome models, for which a polynomial solution is known!

The solution is actually fairly simple: For circular genomes, we take a complete graph on extremities and assign weight $w(x, y) =$ number of genomes containing adjacency xy. For mixed genomes, we add a telomere vertex T_x for each extremity x; T_x will only be connected with x by edge of weight $w(x, \mathrm{T}_x) = 1/2 \cdot$ (number of genomes containing the telomeric adjacency $x\mathrm{T}_x$). The maximum weight perfect matching in this graph then defines the optimal median. This result may be easily generalized to computing median of more than 3 species. In Section 8.3, we show a more efficient algorithm.

3.2 DCJ and Reversal Medians

3.2.1 Complexity

Bad news first. All the studied variants of DCJ and reversal median are NP-hard, even APX-hard[1]. The results are proved by reduction from the Breakpoint-Graph-Decomposition problem, which is intimately related to unsigned sorting by reversals.

Breakpoint graph decomposition. We say that a graph is *bicoloured* if all its edges are coloured either red or blue; we say that a bicoloured graph is *balanced* if all the vertices have degree 2 or 4, every vertex is incident to the same number of red and blue edges, and there is no cycle formed by only red or only blue edges. Caprara (1999) proved that balanced graphs are exactly the breakpoint graphs[2] of some permutations.

A cycle is *alternating*, if it has even length and red and blue edges alternate along the cycle.

Problem 7 (Breakpoint graph decomposition). *Given a balanced graph G, partition the edges of G into a maximum number of edge-disjoint alternating cycles.*

Caprara (1999) proved the NP-hardness of this problem and Berman and Karpinski (1999) strengthened the result by proving its APX-hardness.

Caprara (2003) first noticed the connection between the median problem and breakpoint graph decompositions and proved NP-hardness of reversal and unichromosomal DCJ median. Using his techniques, Tannier et al. (2009) proved NP-hardness for other variants of the DCJ model.

The reduction from breakpoint graph decomposition to DCJ median problem and the idea of the proof are depicted in Fig. 3.2. Let n_2 and n_4 be the number of vertices of degree 2 and 4, respectively. It can be shown that there exists a genome M such that $S(M) \leq n_2 + 3n_4 - k$ if and only if there exists at least k edge-disjoint alternating cycles in G.

[1] namely, the DCJ median problem is APX-hard regardless of whether we use a unichromosomal or a multichromosomal, linear, circular, or mixed variant; reversal median is NP-hard; RT-median was not studied and is conjectured to be NP-hard

[2] in the original definition, the breakpoint graph $BG(\pi, \gamma)$ did not contain 2-cycles corresponding to common adjacencies; each edge corresponded to a breakpoint in π or γ – hence the name "breakpoint graph"

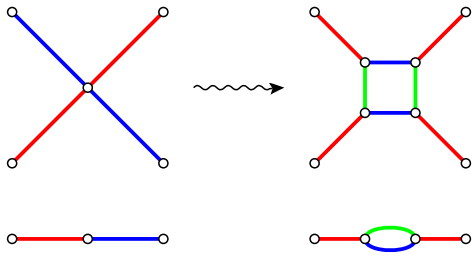


Figure 3.2: Reduction from breakpoint graph decomposition to DCJ median problem. Every vertex v of a balanced graph with its incident edges (left) is replaced by a subgraph on the right. It can be shown (see the adequate subgraph theory in the next section) that any median can be modified to a median not containing red edges; choosing either green or blue edges for DCJ median corresponds to forming alternating cycles from top and bottom edges versus left and right edges.

For unichromosomal DCJ, we need to prove that there is a base matching B such that $\pi_i \cup B$ form a Hamiltonian cycle. For the reversal model, the genomes are further modified to ensure that for median M the distances $rev(M, \pi_i)$ are equal to $dcj(M, \pi_i)$; the hardness result then follows.

3.2.2 Exact Algorithms

Branch and Bound

Siepel's median solver. A branch and bound algorithm for reversal median was proposed by Siepel and Moret (2001). The algorithm uses a more general version of the lower bound based on triangle inequality: Let π_1, π_2, and π_3 be the input permutations such that π_2 and π_3 are separated by distance $d_{2,3}$, and let M be one of their medians. Let ϕ be another permutation on some optimal path from π_1 to some median. Let ϕ be separated from π_1, π_2, and π_3 by distances $d_{1,\phi}$, $d_{2,\phi}$, and $d_{3,\phi}$, respectively. Then the median score obeys these bounds:

$$L^\phi = d_{1,\phi} + \left\lceil \frac{d_{2,\phi} + d_{3,\phi} + d_{2,3}}{2} \right\rceil \leq S^* \leq d_{1,\phi} + \min\Big\{(d_{\phi,2} + d_{\phi,3}), (d_{2,\phi} + d_{2,3}), (d_{3,\phi} + d_{3,2})\Big\} = U^\phi$$

In other words, if we denote S^ϕ the minimum median score of some permutation M' such that ϕ lies on an optimal path from π_1 to M' ($d(\pi_1, \phi) + d(\phi, M') = d(\pi_1, M')$), then for all ϕ

$$L^\phi \leq S^* \leq S^\phi \leq U^\phi$$

and for ϕ on an optimal path from π_1 to some median, $S^* = S^\phi$.

Without loss of generality, let π_1 have the lowest median score from π_1, π_2, and π_3. We start with $\phi = \pi_1$ and try to trace a path from π_1 to some median M. In each step, we will have a set of candidates for ϕ and we pick the "most promising" one. We find all of its as-yet-unvisited neighbours (obtained by a single reversal), which are farther from π_1 and add them to the set of candidates.

Siepel's algorithm first computes the "global" upper and lower bounds U, L for the median score. Furthermore, for each solution ϕ, we keep its own upper and lower bounds U^ϕ, L^ϕ. Solutions can be

pruned when their best possible scores exceed the current global upper bound ($U \leq L^{\phi}$). The upper bound is lowered when a solution is found that has a lower upper bound.

The search ends when a score equal to the global lower bound is found or when no vertex in the queue has a best-possible score lower than the upper bound.

An interesting subproblem in this approach is, given permutations π, to list all optimal reversals (i.e., such that $rev(\pi \circ \rho) = rev(\pi) - 1$). Note that some permutations may have $\Omega(n^2)$ sorting reversals. Siepel and Moret (2001) described an algorithm finding all such reversals in $O(n^3)$. Recently, Swenson et al. (2011) devised a quadratic algorithm, provided that π does not contain bad components. It is an open problem, whether a more efficient output sensitive algorithm exists (running in, say $O(k) + o(n^2)$, where k is the number of sorting reversals).

Caprara's median solver. An alternative branch and bound algorithm was proposed by Caprara (2003) for the unichromosomal DCJ distance; Zhang et al. (2008, 2009) implemented a median solver for RT and DCJ distance based on the same idea.

The main idea is to construct the median by gradually fixing its adjacencies. Let π_1, π_2, and π_3 be the input genomes and suppose we fix adjacency pq of the median M. If necessary, apply a single DCJ operation to get genomes π'_1, π'_2, and π'_3, which contain the adjacency pq. Let m be the number of DCJ operations needed ($0 \leq m \leq 3$) and let $d'_{i,j}$ be the distances between the resulting genomes; then

$$m + \left\lceil \frac{d'_{1,2} + d'_{2,3} + d'_{3,1}}{2} \right\rceil \leq S^{pq},$$

where S^{pq} is the best median score for a solution containing adjacency pq.

To speed up the search, in the first phase, we search for a perfect median, i.e., M such that $S(M) = L = \left\lceil \frac{d_{1,2}+d_{2,3}+d_{3,1}}{2} \right\rceil$; if the input genomes do not have perfect median, we know that $S(M) \geq L + 1$ and we search for genome reaching the new lower bound; if none exists, $S(M) \geq L + 2$, etc. This is faster when the medians are close to perfect.

Decomposition of Multiple Breakpoint Graph

The Caprara's branch and bound algorithm was substantially improved by Xu and Sankoff (2008) using their decomposition theory. We will assume the DCJ model and genomes consisting of (possibly multiple) circular chromosomes.

Multiple breakpoint graph. Recall that we can think of circular genomes as matchings of the extremities. We define a *multiple breakpoint graph* $MBG(\pi_1, \pi_2, \ldots, \pi_k)$ to be a coloured multigraph consisting of the given k matchings, each having its own colour. Note that breakpoint graph $BG(\pi)$ defined in Section 2.3 is a MBG for two genomes – π (with black edges) and identity (with grey edges).

Let π_1, π_2, and π_3 be the input genomes, corresponding to, say red, green, and blue matchings in the breakpoint graph $G = MBG(\pi_1, \pi_2, \pi_3)$. Recall that the DCJ distance between two circular genomes π_i

and π_j is $dcj(\pi_i, \pi_j) = n - c_{i,j}$, where n is the number of markers and $c_{i,j}$ is the number of cycles in $\pi_i \cup \pi_j$.

Let M be another genome with a black matching. Adding edges of M to G results in a so called *median graph* $MBG(M, \pi_1, \pi_2, \pi_3)$. The median score of M is

$$S(M) = dcj(M, \pi_1) + dcj(M, \pi_2) + dcj(M, \pi_3) = 3n - (c_1(M) + c_2(M) + c_3(M)),$$

where $c_i(M)$ is the number of cycles in $M \cup \pi_i$.

Thus, we may reformulate the DCJ median problem as follows: We are given a coloured multigraph G consisting of three matchings (each has edges of different colour) and we seek another matching, such that $c = c_1 + c_2 + c_3$ is maximized.

Decomposers. By the term *subgraph* of an MBG G, we will mean an *induced* proper subgraph of G with an *even* number of vertices. Its *size* will be half the number of its vertices. Let H be such a subgraph.

The edges in $E(H, \overline{H})$, i.e., having one endpoint in H and the other outside H, are called *H-crossing* edges. We say that a matching is *H-crossing* if it contains at least one H-crossing edge.

Graph H is called a *decomposer* if for any MBG G containing it, there is an optimal matching that is not H-crossing. It is a *strong decomposer* if for any MBG G containing it, no optimal matching is H-crossing. Thus, if we find a decomposer H in G, we can decompose our problem into finding an optimal matching in H and an optimal matching in $\overline{H}$.

Note that for genomes with n markers G has $2n$ vertices and $(2n-1)!! = \frac{(2n-1)!}{2^n n!}$ matchings. If H has $2m$ vertices, after the decomposition, the search space is reduced to only $(2m-1)!!(2(n-m)-1)!!$ matchings. Furthermore, it is often not necessary to try all the optimal matchings of H. A minimal set of optimal matchings of H which guarantees that at least one of them must appear in an optimal matching of any MBG G, is called a *major set* of H.

The idea is to build a repertoire of small decomposers (together with their major sets) and to search them in the MBG G and thus reduce the problem.

Adequate graphs. If M is a (perfect) matching on H, we denote $c_{M,i}(H)$ the number of cycles in $M \cup \pi_i$, and $c^*(H)$ the maximum number of cycles $c_{M,1}(H) + c_{M,2}(H) + c_{M,3}(H)$ for subgraph H over all matchings M.

We say that graph H of size m (with $2m$ vertices) is *adequate*, if $c^*(H) \geq 3/2m$; it is *strongly adequate*, if the given inequality is strict.

Theorem 8 (Xu and Sankoff (2008)). *Any adequate graph is a decomposer. Any strongly adequate graph is a strong decomposer.*

A (strongly) adequate graph H is *simple* if it does not contain another (strongly) adequate subgraph as an induced subgraph. All the simple adequate graphs of size 1, 2, and 4 were found by exhaustive search and they are shown on Fig. 3.3.

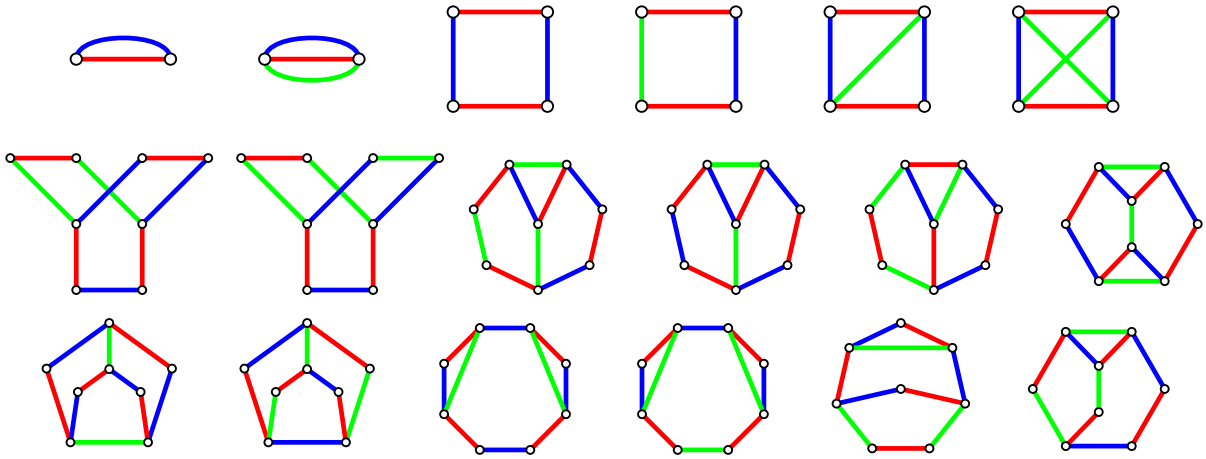


Figure 3.3: Repertoire of all simple adequate graphs of size 1, 2, and 4.

Xu (2009) proved that there are infinitely many simple adequate graphs. If H is a simple adequate graph, all its vertices have degree 2 or 3. Except for size 1, all the *simple* adequate graphs have even size m and $c^*(H)$ is exactly $3/2m$.

Algorithm. The algorithm `ASMedian` is in fact an extension of Caprara's branch and bound algorithm where the median is constructed by gradually fixing its edges (each time, all the possible edges going from one vertex are tried). To speed up the algorithm, in each step we search for an adequate subgraph. If we find one, we fix the edges to one of its optimal matching (we try all the matchings in the major set); if none is found, we resort to trying all possible edges from one vertex.

A clean way of fixing some edges of the resulting median is to *shrink* these edges. Graph G', the result of shrinking edge e in MBG G, is obtained by deleting the endpoints of e together with edges parallel to e and connecting the adjacent edges with the same colour (see Fig. 3.4).

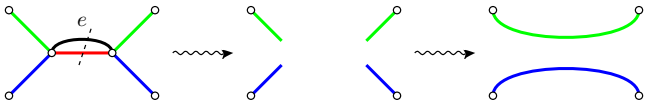


Figure 3.4: Shrinking of the black edge e: delete the endpoints of e together with all parallel edges, connect the adjacent edges with the same colour.

3.2.3 Heuristics

Although the median problem is NP-hard, we would like to use a median solver repeatedly as a quick subroutine in the phylogeny reconstruction and even to compute medians of large genomes (e.g., mammalian genomes have around 25 000 genes). Therefore, fast heuristics are needed.

Good rearrangements. A well known heuristic for median finding is `MGR` (Bourque and Pevzner, 2002; Adam and Sankoff, 2008) based on good rearrangements. We say that a rearrangement operation is *good*, if it moves one genome towards both other genomes. For example, let π_1, π_2, π_3 be the input permutations and let $\pi_1' = \pi_1 \circ \rho$. Reversal ρ is *good* if $d(\pi_1', \pi_2) = d(\pi_1, \pi_2) - 1$ and $d(\pi_1', \pi_3) = d(\pi_1, \pi_3) - 1$. We apply ρ on π_1 and thus reduce the problem to finding median of π_1', π_2, π_3; this way, we move π_1, π_2, and π_3 towards each other in hope that the good reversals preserve medians, i.e. $\mathfrak{M}(\pi_1', \pi_2, \pi_3) \subseteq \mathfrak{M}(\pi_1, \pi_2, \pi_3)$.

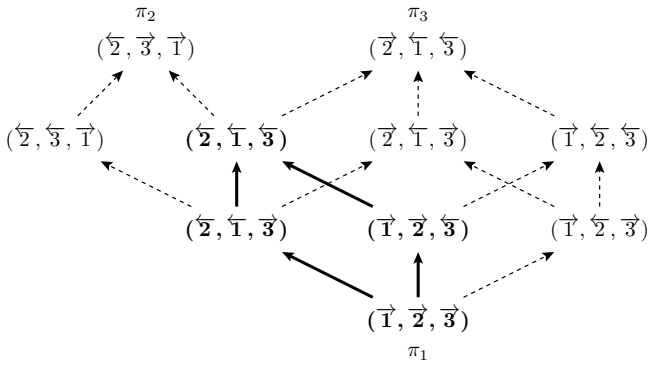
Unfortunately, not all good reversals are median-preserving. Take for example permutations $\pi_1 = (\overrightarrow{1}, \overleftarrow{6}, \overrightarrow{4}, \overleftarrow{3}, \overrightarrow{5}, \overleftarrow{2})$, $\pi_2 = (\overrightarrow{3}, \overrightarrow{1}, \overleftarrow{4}, \overrightarrow{6}, \overleftarrow{5}, \overrightarrow{2})$, $\pi_3 = (\overleftarrow{1}, \overleftarrow{5}, \overleftarrow{4}, \overleftarrow{2}, \overleftarrow{6}, \overrightarrow{3})$, and let ρ be a good reversal which flips the whole genome, so that $\pi_1 \circ \rho = \pi_1' = (\overrightarrow{2}, \overleftarrow{5}, \overrightarrow{3}, \overleftarrow{4}, \overrightarrow{6}, \overleftarrow{1})$. While the best score for the input permutations is $S^*(\pi_1, \pi_2, \pi_3) = 9$, after applying the good reversal ρ, $S^*(\pi_1', \pi_2, \pi_3) = 10$. On the other hand, even though good rearrangements are not necessarily median-preserving, in practice they produce genomes with good median scores.

There are several variants to the basic idea depending on which good rearrangements do we choose, how do we apply them and when do we stop. We can either apply random good rearrangements, or look ahead and greedily choose such good rearrangement, which leaves as many good rearrangements for the next genome as possible. In a "serial" variant, we apply a good rearrangement on π_1 to get π_1', then a good rearrangement on π_2 with respect to π_1' and π_3 to get π_2', and on π_3 with respect to the new genomes π_1' and π_2'. An alternative is a "parallel" variant, where we apply a good rearrangement on π_1, π_2, and π_3 with respect to the original genomes π_1, π_2, and π_3. We can stop when some genome has no good rearrangement, or when none of the genomes has a good rearrangement. Then, we can either output one of the genomes as median, or continue with some "not-so-good" rearrangements until the input genomes meet at one point, which we declare our median. Finally, we may perform a local search and possibly find a better genome in the neighbourhood.

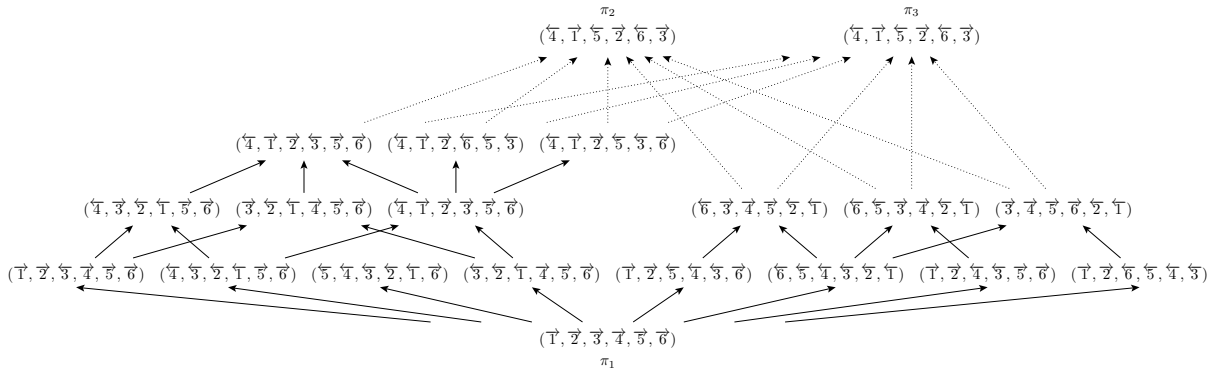
Experiments with reversal model and random permutations indicates that the change in stopping condition and the choice of serial or parallel variant have no significant impact on performance. On the other hand, perhaps surprisingly, heuristics that returned median when no good reversal was available, were on par or even slightly outperformed the traditional solver `MGR` (Bourque and Pevzner, 2002), which resorts to "not-so-good" reversals and performs a local search at the end. The experiments indicate that the expensive local search has little effect on the median score.

Maximal signatures. A more recent and better heuristic to find a median is the method of maximal signatures (Swenson and Moret, 2009). Let $P(\pi, \gamma)$ be the set of all genomes on some optimal path from π to γ. The intersection $P(\pi_1, \pi_2) \cap P(\pi_1, \pi_3)$ is called a *signature* of π_1 with respect to π_2 and π_3, and is denoted $\mathfrak{S}(\pi_1; \pi_2, \pi_3)$, see Fig. 3.5. This is a set of genomes θ such that $d(\pi_1, \pi_2) = d(\pi_1, \theta) + d(\theta, \pi_2)$ and $d(\pi_1, \pi_3) = d(\pi_1, \theta) + d(\theta, \pi_3)$; or equivalently, these are the genomes which are reachable from π_1 using only good rearrangements (with respect to π_2 and π_3).

Maximal signatures are those members of $\mathfrak{S}(\pi_1; \pi_2, \pi_3)$ for which no good rearrangement exists.



(a) All the optimal scenarios transforming π_1 into π_2 and π_3 by reversals; the bold permutations which are both on an optimal path to π_2 and to π_3 constitute the signature $\mathfrak{S}(\pi_1;\pi_2,\pi_3)$. The bold arrows going from π_1 are the good reversals; $(\overleftarrow{2}, \overleftarrow{1}, \overleftarrow{3})$ is the maximal signature.



(b) A slightly more complicated example; only the permutations from signature $\mathfrak{S}(\pi_1;\pi_2,\pi_3)$ are shown; the arrows going from π_1 are the good reversals, the topmost permutations are maximal signatures, for which no good reversal exists. The dotted arrows represent several reversals from the rest of the scenarios. Notice that not all maximal signatures are equally far from π_1; only the 3 permutations on the left are maximum signatures.

Figure 3.5: Two examples of signatures.

These are the "last" genomes that are common to sorting paths from π_1 to both π_2 and π_3 and can be easily found. On the other hand, *maximum signatures* are the maximal signatures, with the maximum distance from π_1 (see Fig. 3.5(b)). The complexity of finding maximum signatures is unknown.

Rajan et al. (2010) suggested a heuristic based on maximal signatures instead of good rearrangements: find maximal signature of each genome with respect to the other genomes and return the best solution. Experiments on simulated data show that the maximal signature heuristic consistently outperforms heuristics based on good rearrangements.

Multiple breakpoint graph decomposition. The most recent and most promising heuristic median solver is based on the decomposition theory of Xu and Sankoff (2008) described in the previous section. Recall that the `ASMedian` algorithm has a repertoire of adequate graphs, which are searched in the input MBG G. If an adequate subgraph is found in G, we can safely fix some edges of the median, however, when G contains no adequate subgraph, `ASMedian` resorts to exhaustive search.

Heuristic by Rajan et al. (2010) searches for adequate subgraphs, but avoids the exhaustive search by fixing some adjacency of the median heuristically. They define *adequacy* of a subgraph as $c - 3/2m$, where c is the number of alternating cycles and m is the number of edges. They propose to choose an edge that maximizes the adequacy of the solution built so far.

Note that the resulting median may be multichromosomal even though the inputs were unichromosomal. If we seek a unichromosomal median, the result is post-processed and small extra circular chromosomes are incorporated into the largest chromosome greedily, increasing the median score as little as possible.

The `ASM` heuristic clearly dominates both `MGR` and maximal signature heuristic in terms of both accuracy and running time; it constitutes the current state of the art median solver.

Chapter 4

Whole Genome Duplication and Halving Problems

4.1 Introduction

Whole genome duplication (WGD) is a rare mutation which dramatically changes the genome architecture: the size of the genome doubles and each gene gains two copies. In this chapter, we study various problems motivated by a WGD. While the general question remains the same:

> Can we reconstruct the ancestral gene order?

the duplicated genomes introduce some new twists and complications to the problems.

Let us first define more precisely, what do we mean by a duplicated and a perfectly duplicated genome. Then we define a distance between such genomes and formally state the halving and guided halving problems which are studied in the sections that follow.

Representing duplicated genomes. Consider a genome where each gene has possibly multiple copies called *paralogs*. For each gene g with k copies, we may label the paralogs by $g_1, g_2, \ldots, g_k$ and then represent the genome as in Section 2.1, treating each copy as a different marker. We call such representation δ a *differentiated* genome.

However, since the labeling of paralogs was completely arbitrary, we consider two genomes that differ only in the subscripts of some genes as equivalent. A single genome with duplicated genes thus corresponds to the equivalence class $[\delta]$ of all differentiated genomes obtained from δ by relabeling the paralogs.

In this chapter, we consider mainly *duplicated* genomes that underwent a single whole genome duplication and thus have exactly two copies of each gene. If g is one copy, we denote the other paralogous copy by $\bar{g}_i$ (so $\bar{g}_1 = g_2$ and $\bar{g}_2 = g_1$). Similarly, if p is an extremity, we will denote by $\bar{p}$ the paralogous

extremity and if $x = pq$ is an adjacency (possibly telomeric), then $\bar{x} = \bar{p}\bar{q}$ will denote the paralogous adjacency.

Doubled genomes. We say that a duplicated genome immediately after the WGD is *perfectly duplicated* or simply *doubled*. Formally, we say that a differentiated genome θ is *doubled*, if for each adjacency pq in θ, adjacency $\bar{p}\bar{q}$ is also in θ and $p \neq \bar{q}$. This is the same as saying that if we ignore the subscripts (1's and 2's), every linear chromosome has an identical copy and every circular chromosome has either an identical copy, or is itself composed of two successive identical copies (see Fig. 4.1).

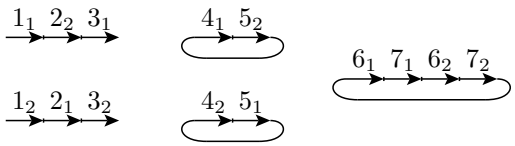


Figure 4.1: Example of a doubled genome.

If α is an ordinary (not duplicated) genome, the pre-duplication ancestor, we denote by $\alpha \oplus \alpha$ the set of all (differentiated) perfectly duplicated genomes obtained by doubling of α.

Distance between duplicated genomes. If δ and γ are two differentiated genomes, we can compute the genomic distance as in Chapter 2. How do we compute the distance between two duplicated genomes $[\delta]$ and $[\gamma]$? Note that we do not know which copy of g in $[\delta]$ corresponds to which copy of g in $[\gamma]$.

Applying the parsimony principle again, we may define the distance between two duplicated genomes $[\gamma]$ and $[\delta]$ as the minimum distance between differentiated genomes $\gamma' \in [\gamma]$ and $\delta' \in [\delta]$. In fact, we can fix one $\gamma' \in [\gamma]$ and take the minimum over $\delta' \in [\delta]$.

Similarly, we can define the distance between an ordinary genome α and a duplicated genome $[\delta]$, also called *double distance* and denoted $dd(\alpha, [\delta])$ as the minimum distance between $\theta \in \alpha \oplus \alpha$ and $\delta' \in [\delta]$. Again, without loss of generality, we can fix one $\delta' \in [\delta]$ and find the minimum over $\theta \in \alpha \oplus \alpha$.

Problem 8 (Double distance). *Given a duplicated genome $[\delta]$ and an ordinary genome α, compute $dd(\alpha, [\delta])$.*

Genome halving. Now we are ready to define precisely the halving problem: Imagine genome α that underwent a whole genome duplication and then evolved into genome $[\delta]$ (see Fig. 4.2). We are given $[\delta]$ and our goal is to reconstruct genome θ right after the whole genome duplication (or equivalently, the ancestral genome α).

Problem 6' (Genome Halving). *Given a (differentiated) duplicated genome δ, find a doubled genome θ such that $d(\theta, \delta)$ is minimal. Equivalently, the problem is to find the pre-duplication ancestor α such that the double distance $dd(\alpha, [\delta])$ is minimal. This distance is called the* halving distance *of $[\delta]$, written $h(\delta)$.*

We will study the halving problem in the following section.

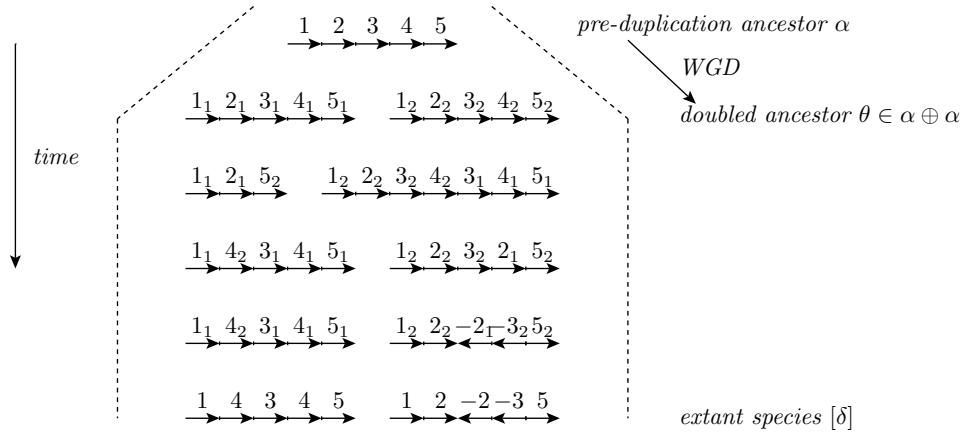


Figure 4.2: The ancestral genome undergoes a whole genome duplication and subsequently evolves by rearrangements. The goal of genome halving is to reconstruct the genome immediately after the whole genome duplication, given the gene order of the extant species.

Guided genome halving. Genome halving is a well-defined optimization problem, which can be moreover solved in polynomial time in many interesting cases, as we will see in the next section. The only drawback is that, usually, there are many different optimal solutions (Seoighe and Wolfe, 1998).

To remedy this situation, we can use the approach inspired by the median problem: In the median problem, we try to reconstruct the common ancestor α of two species by considering a third, outgroup species and then minimizing the sum of distances from α. Similarly, when we are trying to reconstruct the pre-duplication ancestor α of $[\delta]$, we may consider an ordinary (not duplicated) outgroup ρ and minimize the sum of distances from α. This is the so called guided halving problem:

Problem 9 (Guided halving). *Given a duplicated genome $[\delta]$ and an ordinary genome ρ, find an ordinary genome α that minimizes*

$$S(\alpha) = d(\rho, \alpha) + dd(\alpha, [\delta]).$$

We study the guided halving problem in Section 4.3.2. The known results on the halving and guided halving problems are summarized in Table 4.1.

4.2 Halving Problems in the Breakpoint Model

The halving problems in the breakpoint model were studied only recently by Zheng et al. (2008) and Tannier et al. (2009).

General breakpoint model. In the general breakpoint model, we are allowed to have multiple linear or circular chromosomes. Recall, that the breakpoint distance is defined as

$$bp(\pi, \gamma) = n - a(\pi, \gamma) - \frac{e(\pi, \gamma)}{2},$$

Table 4.1: Complexity of halving and guided halving in various genome models. The unichromosomal models may be linear or circular, the multichromosomal models may be circular or mixed. References: [a] Section 8.2, [b] Section 8.3, [c] Tannier et al. (2009), [d] Zheng et al. (2008), [e] Alekseyev and Pevzner (2007c), [f] Section 6.3, [g] Mixtacki (2008), [h] El-Mabrouk and Sankoff (2003), however, the result contains flaws.

Genome Model		Halving	Double Distance	Guided Halving
breakpoint	unichromosomal	NP-hard [a]	$O(n)$	NP-hard [d]
	multilinear	NP-hard [a]	$O(n)$	NP-hard [d]
	multichromosomal	$O(n^3)$, $O(n)$ [b]	$O(n)$ [c]	$O(n^3)$, $O(n\sqrt{n})$ [b]
DCJ	unichromosomal	$O(n)$ [e]	open (not studied)	
	multilinear	$O(n)$ [f]		
	multichromosomal	$O(n)$ [g]	NP-hard [c]	NP-hard [c]
RT	unichromosomal	$O(n^2)$ [e]	open (not studied)	
	multilinear	polynomial? [h]		

where $a(\pi, \gamma)$ is the number of adjacencies and $e(\pi, \gamma)$ the number of telomeres that π and γ have in common.

The double distance can be computed similarly (Tannier et al., 2009):

$$dd_{bp}(\pi, [\delta]) = 2n - a(\pi, [\delta]) - \frac{e(\pi, [\delta])}{2},$$

where $a(\pi, [\delta])$ is the sum, for every adjacency xy in π, of the number of adjacencies among $x_1y_1, x_1y_2, x_2y_1, x_2y_2$ in δ. In other words, we say that π and $[\delta]$ have adjacency xy in common, if x, y are adjacent in π and x_i, y_j are adjacent in δ for some i and j. We say that they have the adjacency xy twice in common, if either x_1y_1 and x_2y_2, or x_1y_2 and x_2y_1 are adjacent in δ. Then $a(\pi, [\delta])$ is the number of adjacencies in common and $e(\pi, [\delta])$ the number of telomeric adjacencies in common, where adjacencies twice in common are counted as 2.

Tannier et al. (2009) also showed that the halving problems are easily solved in cubic time. The algorithms are similar to the algorithm for breakpoint median we reviewed in Section 3.1.2.

For the halving problem, we take a complete graph on extremities and assign weight $w(x, y) =$ number of adjacencies among $\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}$ in δ. For mixed genomes, we add a telomere vertex T_x for each extremity x and an edge $x\mathrm{T}_x$ of weight $w(x, \mathrm{T}_x) = 1/2 \cdot$(number of adjacencies among $\{x_1\mathrm{T}_{x_1}, x_2\mathrm{T}_{x_2}\}$ in δ. Thus, the weights in this graph are $w(x, y) \in \{0, 1, 2\}$ and $w(x, \mathrm{T}_x) \in \{0, \frac{1}{2}, 1\}$. The maximum weight perfect matching in this graph defines the pre-duplication ancestor.

Similarly, for the guided halving problem, we take the same graph and weights $w(x, y) =$ number of adjacencies among xy in ρ and $\{x_1y_1, x_1y_2, x_2y_1, x_2y_2\}$ in δ and $w(x, \mathrm{T}_x) = 1/2 \cdot$ (number of adjacencies among $x\mathrm{T}_x$ in ρ and $\{x_1\mathrm{T}_{x_1}, x_2\mathrm{T}_{x_2}\}$ in δ. In this case, the weights are $w(x, y) \in \{0, 1, 2, 3\}$ and $w(x, \mathrm{T}_x) \in$

$\{0, \frac{1}{2}, 1, \frac{3}{2}\}$ and again, the maximum weight perfect matching in this graph defines the optimal pre-duplication ancestor.

We show more efficient algorithms for the halving and guided halving problems advertised in Table 4.1 in Section 8.3.

Multilinear and unichromosomal models. In the multilinear and unichromosomal models, where the number and type of chromosomes is restricted, the guided halving problem is NP-hard. This was shown by (Zheng et al., 2008) using a reduction from the breakpoint median problem that we have studied in Section 3.1.

Tannier et al. (2009) state that the double distance formula for the general breakpoint model remains valid in the multilinear model. They also conjectured that the halving problem is tractable – after all, it can be solved in linear time in much more complicated models. Alas, in Section 8.2, we show the halving problem in multilinear and unichromosomal models is NP-hard.

4.3 Halving Problems in the DCJ Model

4.3.1 Halving

In this section, we study the genome halving problem under the DCJ model. We may treat the halving problems in other models (such as reversal, RT, or multilinear DCJ) as restricted versions of DCJ halving.

Historically, the research on the halving problem started with the most complicated reversal-translocation model. The problem was introduced by El-Mabrouk and Nadeau (1998) and studied in a series of papers by El-Mabrouk et al. (1999); El-Mabrouk and Sankoff (1999b, 2003). This effort culminated in a paper by El-Mabrouk and Sankoff (2003) announcing a linear-time algorithm for genome halving in the RT-model, which was entitled "one of the most technically challenging results in computational biology" by Alekseyev and Pevzner. Its proof spans almost 40 pages.

Unfortunately, the result is not entirely correct: the first flaw was noticed by Alekseyev and Pevzner (2007c); also note that the result was based on Tesler's formula for RT-distance which was itself flawed, as we discussed in Section 2.4.

Alekseyev and Pevzner (2007b) gave a quadratic-time algorithm in the (circular) reversal model and Alekseyev and Pevzner (2007a) generalized the result to multichromosomal circular RT, DCJ, and 3-break models.

DCJ halving. In the much simpler DCJ model, the halving problem was solved by Warren and Sankoff (2009b) and Mixtacki (2008) (which corrected a small error and simplified the result).

The halving distance, a perfectly duplicated ancestor and one particular halving scenario can be calculated using a concept similar to an adjacency graph from Section 2.2 – a so called natural graph $NG(\delta)$ (El-Mabrouk and Sankoff, 2003). Vertices of this multigraph are adjacencies of δ, and two

adjacencies are connected by an edge, if they share a paralogous extremity. The natural graph consists of paths and cycles only, and θ is perfectly duplicated if and only if $NG(\theta)$ consists of 2-cycles and 1-paths only.

In fact, we may consider the adjacency graph as a special case of a natural graph: if π and γ are two ordinary genomes and we denote $\pi_1 \oplus \gamma_2$ a duplicated genome obtained as a union of π and γ where genes from π are labeled by 1 and genes from γ are labeled by 2, then $AG(\pi, \gamma) = NG(\pi_1 \oplus \gamma_2)$. In this curious way, we may view the sorting problem as a restricted halving problem, where we are only allowed to rearrange one half of the genome.

Our analysis from Section 2.2 can be easily adapted to the case of halving: Let c_e be the number of even cycles and p_o the number of odd paths in the natural graph $NG(\delta)$. Since a single DCJ operation can only change the number of even cycles by at most 1 or the number of odd paths by at most 2, we immediately obtain a lower bound on the halving distance

$$h_{dcj}(\delta) \geq n - (c_e + \lfloor p_o/2 \rfloor).$$

(Compare this formula with Corollary 1 on page 33; the missing floor function is not need there, since adjacency graphs are bipartite and contain even number of odd-length paths.)

Similarly, it is easy to prove that there is always a DCJ operation that increases either the number of even cycles or the number of odd paths by 2 so the lower bound is tight (see an example on Fig. 4.3).

Theorem 9 (Mixtacki (2008)). *Let δ be a duplicated genome with $2n$ markers. The minimal distance between δ and any doubled genome is*

$$h(\delta) = n - (c_e + \lfloor p_o/2 \rfloor),$$

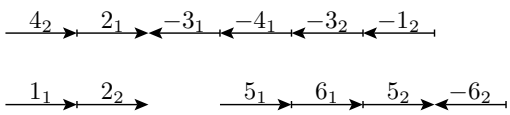
where c_e is the number of even cycles and p_o the number of odd paths in the natural graph $NG(\delta)$.

4.3.2 Guided Halving

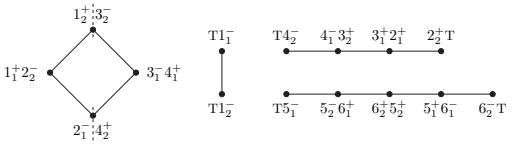
Shortly after the halving problem was introduced, Seoighe and Wolfe (1998) noted the extreme non-uniqueness in the solution space and suggested using outgroup species to "guide" the reconstruction of the pre-duplication ancestor.

The guided halving problem was first tackled computationally by Zheng et al. (2006). However, they proposed only a heuristic solution: exhaustively enumerate or just sample the space of optimal halving solutions and starting from these, do a local search trying to minimize the guided halving score. In fact, Zheng et al. (2006) only considered optimal paths from some halving solution α and outgroup ρ. This seemed acceptable for the short genomes they were working with.

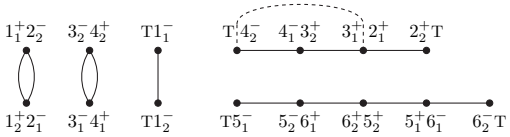
In their follow-up paper, Zheng et al. (2008) proposed a much faster greedy heuristic. They also considered the problem of genome halving with two (or multiple) outgroups guiding the ancestral reconstruction and applied their software to reconstructing the pre-duplication ancestor of *Saccharomyces*



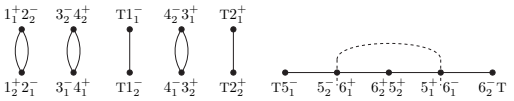
(a) Input genome δ with 6 duplicated genes.



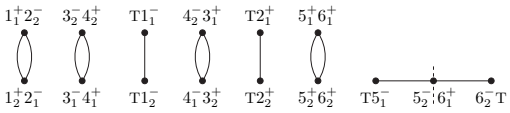
(b) The natural graph $NG(\delta)$ contains 1 even cycle and 2 odd paths so $h_{dcj}(\delta) = 6 - (1 + \lfloor 2/2 \rfloor) = 4$. Step 1: Cut $1_2^+3_2^-$, $2_1^-4_2^+$ and join $1_2^+2_1^-$, $3_2^-4_2^+$.



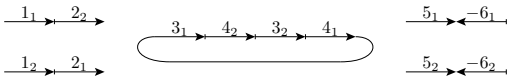
(c) Step 2: Cut $\mathrm{T}4_2^-$, $3_1^+2_1^+$ and join $\mathrm{T}2_1^+$, $4_2^-3_1^+$.



(d) Step 3: Cut $5_2^-6_1^+$, $5_1^+6_2^-$ and join $5_1^+6_1^+$, $5_2^-6_1^-$.



(e) Step 4: Cut $5_2^-6_1^+$. The natural graph after this step will consist only of 2-cycles and 1-paths so the genome will be perfectly duplicated.



(f) The reconstructed perfectly duplicated ancestral genome.

Figure 4.3: Example of DCJ halving. Genome δ (top left) is transformed into a perfectly duplicated genome (bottom right) using 4 DCJs.

cerevisiae and *Candida glabrata* guided by the genomes of three other yeasts that diverged before the WGD event.

The hope for an exact efficient algorithm was shattered by Tannier et al. (2009). They proved that not only the guided halving problem, but even computing the DCJ double distance is NP-hard. Thus, even computing the guided halving score for an arbitrary genome α is hard.

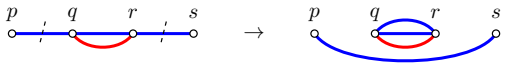
Interestingly, in the DCJ model, the halving problem can be solved in linear time, even though the double distance, the very function we try to optimize, is NP-hard to compute. This is not a contradiction since in the halving problem, we just compute the minimum over all genomes, we do not need to compute the double distance for the inputs constructed to be hard.

The proofs of Tannier et al. (2009) are similar to proving hardness of the median problem and use the techniques of Caprara (2003), which we discussed in Section 3.2. These arguments seem to go through for the more complicated models such as reversal or RT, so double distance and guided halving is conjectured to be NP-hard in these models too.

Inspired by the progress on the median problem, Gavranović and Tannier (2010) proposed a new lower bound for guided genome halving: Recall, that the median score can be lower bounded by

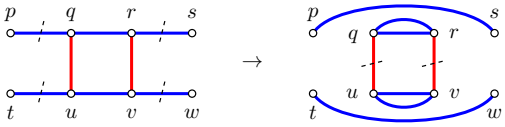
$$S(M) \geq \frac{d(\pi_1, \pi_2) + d(\pi_2, \pi_3) + d(\pi_3, \pi_1)}{2}$$

– a result that easily follows from the triangle inequality. Similarly, the guided halving score can be lower



(a) We obtain three parallel edges by cutting pq, rs and joining ps, qr.

(b) Here, we obtain 3 parallel edges connecting qt and 2 parallel edges connecting rs using 2 DCJ operations: $pq, tu \to pu, qt$ and $qr, st \to qt, rs$.



(c) In this case, using 2 DCJ operations ($pq, rs \to ps, qr$ and $tu, vw \to tw, uv$), we obtain two pairs of parallel blue edges forming a double-blue/red cycle of length 4 (the result on the right). This is followed by a DCJ operation on the red edges ($qu, rv \to qr, uv$) forming two triple edges.

Figure 4.4: The subgraphs identified by the algorithm of Gavranović and Tannier (2010) and their resolution (in order of precedence).

bounded by

$$S(\alpha) \geq \frac{dd(\rho, [\delta]) + h(\delta)}{2}.$$

A little catch here is that for example for DCJ, computing the double distance $dd(\rho, [\delta])$ is NP-hard. Gavranović and Tannier (2010) therefore suggest to use a looser lower bound replacing $dd_{dcj}(\rho, [\delta])$ by $dd_{bp}(\rho, [\delta])/2$.

Gavranović and Tannier (2010) also designed a more precise heuristic algorithm: They use the notion of a contracted breakpoint graph[1] $CBG(\rho, [\delta])$ where vertices are extremities of ρ and vertices p, q are connected by a red edge if pq is an adjacency in ρ, and by $k \in \{0, 1, 2\}$ blue edges if there are k adjacencies of the form $p_i q_j$ in δ. Finding an optimal pre-duplication ancestor is then equivalent to finding the shortest sequence of DCJ operations which transforms this graph into a graph where all connected vertices are connected by three parallel edges.

The algorithm is inspired by the theory of adequate subgraphs by Xu and Sankoff (2008) (see Section 3.2.2). It tries to identify certain subgraphs with known optimal solution in the contracted breakpoint graph (see Fig. 4.4). If no such pattern is found, the algorithm greedily chooses a random optimal halving rearrangement.

Unfortunately, Gavranović and Tannier (2010) found only a few subgraphs and developed no underlying theory comparable to the theory of decomposers by Xu and Sankoff (2008).

[1] this is an analogy of a multiple breakpoint graph we used in Section 3.2.2; it is also a line-graph of the natural graph used by Mixtacki (2008)

4.4 Other Variants

Halving of hybrid genomes. El-Mabrouk and Sankoff (1999a) studied variants of the halving and guided halving problems inspired by hybridization of two different existing species (this is most widespread in the plant kingdom). Imagine two genomes π and γ over two disjoint sets of genes A and B. These genomes hybridize forming a single genome $\theta = \pi \oplus \gamma$ – a union of the chromosomes in π and γ – and then all three species evolve and their genomes are rearranged into the present-day form π', γ', and θ'.

Given the present-day order θ' and sets A and B, can we reconstruct the ancestral genome θ such that all chromosomes in θ contain only genes from one set, while minimizing the distance $d(\theta, \theta')$? Given the present-day gene-orders θ', π', and γ', can we reconstruct the ancestral genomes θ, π, and γ such that $\theta = \pi \oplus \gamma$, while minimizing the sum of distances $d(\pi, \pi') + d(\gamma, \gamma') + d(\theta, \theta')$? El-Mabrouk and Sankoff (1999a) gave a linear-time exact algorithm for the former problem in the RT model and a heuristic solution for the latter.

Aliquoting. A natural generalization of the halving problem, where a genome undergoes a whole genome duplication and each gene has exactly two copies is the *aliquoting* problem. Here, we imagine a genome that undergoes a k-way polyploidization event (e.g., several whole genome duplications), so that each gene has exactly k copies and each chromosome has k perfect copies. We call such a genome a *polyploid*. Again, throughout the evolution, the genes are shuffled over the genome and the problem is to reconstruct the genome just before the polyploidization event such that the number of rearrangements throughout the evolution is minimized.

Warren and Sankoff (2009a) introduced the problem and proposed a heuristic solution. Warren and Sankoff (2011) solved the problem for the general breakpoint model and gave a 2-approximation algorithm in the DCJ model. The problem is conjectured to be NP-hard (for the DCJ and similar models). For $k = 3$ the problem reminds of the classical median problem except that instead of 3 distinct genomes, we have only a single genome and different copies of the ancestral chromosomes may get intermixed.

Chapter 5

Reconstructing Phylogenies

5.1 Introduction

Tree of life. Phylogeny is a reconstruction of evolutionary history of a group of organisms, and is usually represented in a form of a rooted or an unrooted phylogenetic tree, where leaves represent extant species and internal nodes represent their ancestors (see Fig. 5.1).[1]

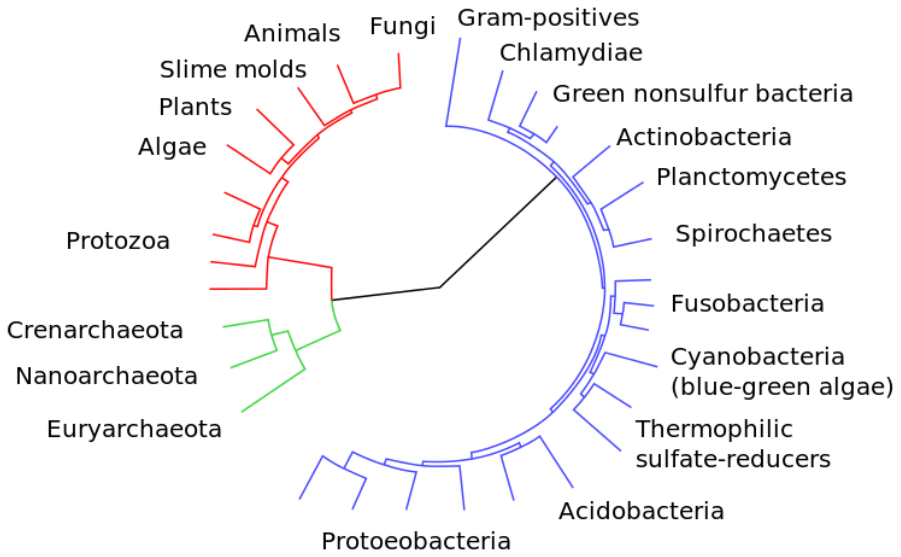


Figure 5.1: Tree of life.

From the times of Charles Darwin, an ambitious project of biologists is to reconstruct the whole

[1] In some cases, for instance among some species of bacteria and archaea, it is possible that the genetic information is exchanged between organisms from different species. This is known as *horizontal gene transfer* and in such case, the phylogeny forms a directed acyclic graph rather than a tree. In this chapter, we will not concern ourselves with such events and we will assume that the phylogeny is indeed a tree.

"tree of life" – the phylogenetic tree of all species. Currently, an ongoing *Tree of Life Web Project* (`http://www.tolweb.org/tree/`) gathers and provides information about phylogeny and diversity of life on Earth.

First phylogenetic tree reconstructions were based on morphological data. These are however hard to quantify, measure, and compare; they are not very informative about more distant species and sometimes they could be even misleading.

With advent of molecular biology, it became possible to reconstruct phylogenetic trees based on genetic sequences – DNA, RNA, or proteins. There are essentially two types of genomic data we can use for phylogeny reconstruction:

1. sequence data, and
2. gene order data.

Since there is considerable overlap between the methods for reconstructing phylogenies from sequence and from gene-order data, we review both data types in the sections to follow.

Phylogenies from sequence data. When reconstructing phylogeny from sequence data, we first have to find segments from different organisms (usually genes) which most likely evolved from a common ancestral sequence. The sequences are assumed to evolve by point mutations, but also by short insertions and deletions (see Fig. 5.2(a)). Therefore, in the next step, we have to align the sequences – add "gap" characters representing inserted or deleted bases so that all sequences have the same length and a single column in the alignment corresponds to the evolution of a single position in the ancestral sequence. We try to achieve this by aligning identical or similar characters while adding as few gaps as possible. Only the third step is the actual phylogeny reconstruction, which is usually done column-by-column, assuming the neighbouring columns evolve independently.

Phylogenies from gene-order data. When reconstructing phylogeny from gene-order data, we first have to find segments (genes or synteny blocks) that all the studied species have in common and determine their order and orientation. If our genome model does not support duplications and deletions, we might have to discard some data. Then we assume the genomes evolved by the rearrangement operations of our model and we try to find the most parsimonious reconstruction of the evolutionary history (see Fig. 5.2(b)).

Sequence data vs. gene-order data. Reconstruction from sequence data has several advantages: Firstly, when working with sequence data, we only need to determine the sequence of one or few genes as opposed to gene orders, where the whole genome is explored. While the sequence data are abundant, this is not the case for gene-order data.

Secondly, the evolution by point mutations and small insertions/deletions is much better understood than rearrangements. We have good probabilistic models for substitutions. In contrast, we do not have

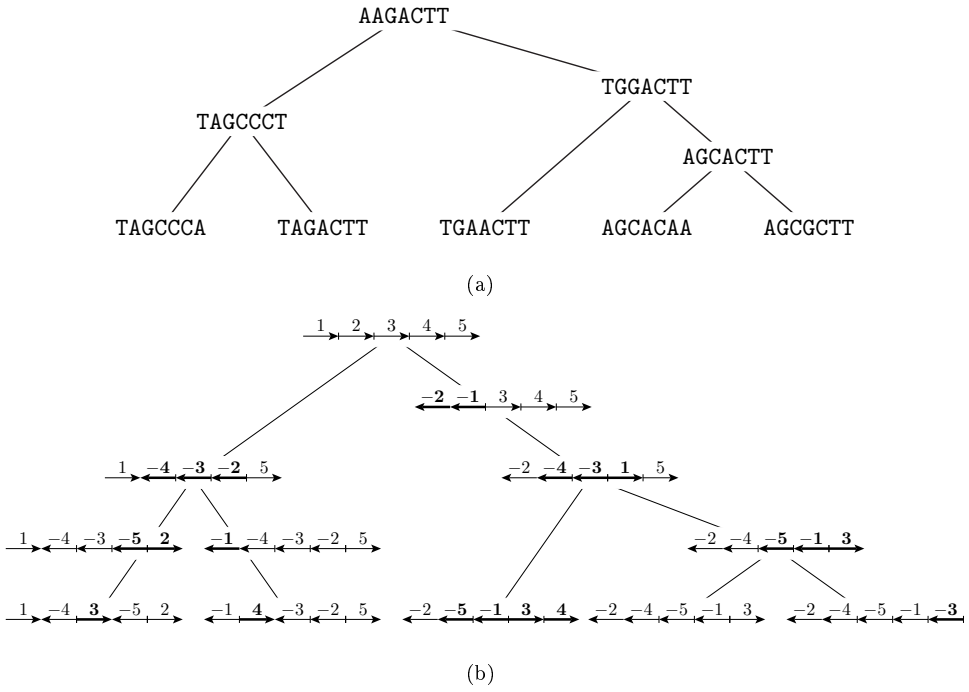


Figure 5.2: Reconstructing phylogeny from (a) sequence and (b) gene-order data. Genomes at leaves are the extant species, while genomes at internal vertices represent the reconstructed ancestral species.

any good probabilistic model of rearrangements. We have strong evidence for reversals, but only some evidence for transpositions, and the relative prevalence of various rearrangement events is unknown. Also it turns out that reversals of different lengths are not equally frequent.

Thirdly, reconstruction from sequence data seems to be easier computationally. Even though reconstructing the phylogenetic tree is NP-hard for both data types, reconstruction of ancestral genomes when the phylogenetic tree is given is solvable in polynomial time for sequence data (see Sections 5.3 and 5.4), while the small phylogeny problem is NP-hard in most rearrangement models even for three genomes (as we have seen in Chapter 3).

The sequence data have disadvantages as well. One serious problem with sequence data is that by analyzing different genes, we may obtain different reconstructed phylogenies. Alas, this is not a bug but a feature: the individual genes may indeed have had quite different evolutionary histories. This discrepancy between the phylogenetic tree of the studied species and trees of individual genes is known as the *gene tree vs. species tree* problem (Page and Charleston, 1997; Maddison, 1997). We can either do a combined analysis where all sequences of one species are concatenated, or a separate analysis followed by reconciliation of the resulting gene trees. In this case, some sort of *consensus tree* is searched (Doyon et al., 2011).

Another disadvantage of sequence data is caused by insertions and deletions – these are handled in the first step by aligning all the sequences. However, alignment of multiple sequences is hard and only poorly solved (Moret, 2005). In contrast, note that there are no problems with alignment and no gene

tree vs. species tree conflicts when working with gene-order data.

Finally, the advantage of gene-order data already mentioned in Chapter 1 is that rearrangement events are much rarer than point mutations in the evolution (Rokas and Holland, 2000). This enables us to look farther in the evolutionary history and to reconstruct phylogenies of far more distant species.

Methods. The phylogeny reconstruction methods can be roughly divided into three categories: distance-based methods, parsimony, and maximum likelihood methods (including Bayesian methods). We study these methods in the sections that follow.

Note that there are two more approaches to phylogeny reconstruction that are beyond the scope of our survey and we do not cover them: the model-free methods and metamethods.

In the *model-free methods*, there are no definitions of rearrangement operations or genomic distance. We simply find features common to the extant species such as common adjacencies, common intervals, or common "near-intervals" and then we try to reconstruct ancestral genomes with such features. These features may be usually written in a matrix form and the problem is formulated using some variant of a consecutive ones problem (see Chauve and Tannier (2008); Gavranović et al. (2011); Ma et al. (2006)).

Since the phylogeny reconstruction is a hard problem, we may try to decompose the set of species into a number of overlapping subsets, use any method discussed in this chapter for each subset, and then combine the resulting phylogenetic trees to produce a phylogeny for the whole dataset. Readers interested in such *metamethods* are referred to the publications of Warnow (2006) and Bininda-Emonds (2004).

5.2 Distance-based Methods

The *distance-based methods* rely solely on the distances between the species and thus can be used with sequence data as well as with gene-order data. These methods only reconstruct the phylogenetic tree, not the ancestral genomes.

One popular strategy is *neighbour joining* (Studier et al., 1988): Given the distances between all pairs of genomes, we construct the tree gradually from leaves bottom-up. In each step, we find a pair of genomes i and j which minimize the value

$$Q(i,j) = (n-2)d_{i,j} - \sum_{k=1}^{n} d_{i,k} - \sum_{k=1}^{n} d_{j,k}$$

where n is the number of species and $d_{i,j}$ is the distance between i^{th} and j^{th} genome. We create their common ancestor k, join i and j with k in the resulting tree, and repeat the procedure with i and j replaced by their ancestor k and distances from k set to $d_{k,\ell} = (d_{i,\ell} + d_{j,\ell} - d_{i,j})/2$.

The main advantage of this approach is that it can be implemented in polynomial time and thus it is feasible even for phylogenies spanning thousands of species. The canonical algorithm runs in $O(n^3)$ in the worst case, however, there are efficient implementations which are much faster on average. Wheeler (2009) describe an implementation named `NINJA` which can scale to inputs larger than 100,000 species.

A desirable feature of the neighbour joining method is that, if given correct distances, it reconstructs the correct tree. More precisely, if the given distances are additive, i.e., there exists a tree with edge lengths such that the distance between two species equals the sum of the edge lengths along the path connecting them, neighbour joining finds this tree. Atteson (1997) proved that this remains true even for "nearly additive" distances and thus the method is statistically consistent under many models of evolution.

Bruno et al. (2000) proposed a variant of neighbour joining called `Weighbor` which improves the accuracy by taking into account the fact that errors in distance estimates are usually much larger for longer distances.

A different phylogeny reconstruction algorithm based on the balanced minimum evolution method was developed by Desper and Gascuel (2004). Their program `FastME` has shown better topological accuracy than other distance based method such as neighbour joining or `Weighbor`.

Note that the accuracy of distance methods depends entirely on the accuracy of the distance estimation. This motivates the study of algorithms for estimating the *true* evolutionary distance (as opposed to the minimum number of rearrangements). We refer the interested reader to the works of Wang and Warnow (2001); Moret et al. (2002a); Lin and Moret (2008). Most recently, Lin et al. (2010) showed how to estimate the true evolutionary distances in the DCJ model with duplications and losses.

5.3 Parsimony Methods

Parsimony methods try to reconstruct both phylogenetic tree and ancestral genomes while minimizing the overall number of mutations.

Sequence data. If we work with sequence data, we can use Hamming distance (or its modification) as a distance measure. Note that given a phylogenetic tree and sequences of the extant species, it is easy to reconstruct the ancestral sequences (minimizing the total Hamming distance on edges of the tree) character by character by simple bottom-up dynamic programming (Fitch and Margoliash, 1967). On the other hand, Foulds and Graham (1982) showed that determining the best tree together with ancestral sequences is NP-hard. Indeed, if the input sequences are binary, the problem is to find a full Steiner tree of minimum length on an n-dimensional hypercube (in the full Steiner tree, all the input vertices have to be its leaves). The reduction is from the Exact-3-Set-Cover problem; see also Day (1983) for other variants of the problem.

Gene-order data. For gene-order data, the phylogeny problems seem to be even harder – even for three extant species, we get the median problem as a special case and we have shown in Chapter 3 that the median problem is NP-hard for most rearrangement distances.

The ancestral genome reconstruction problem under the maximum parsimony criterion is known as the *small phylogeny* problem. More precisely, fix the set of all genomes $\mathcal{G}$ in some model; consider a

phylogenetic tree $T = (V, E)$ with the set of leaves L and genomes of extant species $\pi : L \to \mathcal{G}$. An *evolutionary history* is a function $H : V \to \mathcal{G}$ extending π to the internal (ancestral) vertices. Our goal is to find an evolutionary history H which minimizes the score

$$S_T(H) = \sum_{\{u,v\} \in E} d(H(u), H(v)).$$

$S_T(H)$ is the overall evolutionary distance, the total number of rearrangement operations in history H.

In the *large phylogeny* problem, both the best tree T and the evolutionary history H (minimizing $S_T(H)$) should be reconstructed. We may think of the large phylogeny problem as a variant of the full Steiner tree problem: imagine a graph where $\mathcal{G}$ is the set of vertices and two genomes are connected by an edge, if their distance is 1. The problem is to connect the extant species (the Steiner vertices) by as few edges as possible. Obviously, this is a huge graph given only implicitly and it would be impractical to actually construct it.

Character encodings. One of the approaches to rearrangement phylogeny problems proposed by Cosner (1993) was to encode the gene-order data as sequences, use the algorithms for analysis of sequence data, and finally convert the results back to gene-orders.

One possibility is to use *maximum parsimony on binary encodings (MPBE)* (Cosner, 1993; Cosner et al., 2000). We have a character X_{pq} for each adjacency pq present in at least one of the input genomes and we define $X^k_{pq} = 1$ if genome π_k contains adjacency pq, and 0 otherwise.

Another possibility is to use *maximum parsimony on multistate encodings (MPME)* (Bryant, 2004; Wang et al., 2002; Tang and Wang, 2005). In this encoding, we have characters X_m and X_{-m} for each marker m; value of X^k_m is the (signed) marker immediately following m in genome π_k; X^k_{-m} is defined analogously for the other strand so that $X_\ell = m$ if and only if $X_{-m} = -\ell$. Bryant (2004) showed that MPME can be converted into MPBE by Hamming distance-preserving mapping and the best score under MPME encoding approximates the breakpoint score of the tree better than MPBE.

The problem with character encodings is that the ancestral sequences resulting from sequence analysis may not correspond to any genomes. Cosner et al. (2000) proposed to ignore these sequences, use the tree topology only, and compute the ancestral genomes by other methods. Tang and Wang (2005) attempted to transform the ancestral sequences into valid MPBE encodings of (unichromosomal) genomes while minimizing the Hamming distance. They claim the problem is NP-hard, while the analogous problem for MPME is open.

Steinerization approach to small phylogeny. The prevailing method for reconstructing ancestral genomes (given a phylogenetic tree) is the *Steinerization method* proposed by Blanchette et al. (1997); Sankoff and Blanchette (1998) (we show a new, more general approach in Chapter 7). The idea is to start with some ancestral genomes and then repeatedly replace genomes by medians of genomes in the neighbouring vertices until a local optimum is reached (see Algorithm 1). This optimum may not be

Algorithm 1: Steinerization (Sankoff et al., 1976, 1996)

Data: phylogenetic tree T and initial evolutionary history H

Result: locally optimal evolutionary history

```
repeat
    for v ∈ V − L do
        let π1, π2, π3 be the genomes in the neighbouring vertices;
        compute M ∈ 𝔐(π1, π2, π3);
        if M is better than H(v) then
            replace H(v) by M;
        end
    end
until no improvement ;
return H
```

global but in practice, Steinerization gives good results. The algorithm is implemented in `BPAnalysis` (Blanchette et al., 1997; Sankoff and Blanchette, 1998) and `GRAPPA` (Moret et al., 2001a,b, 2002a).

In `BPAnalysis`, the median solver used is the branch and bound algorithm for travelling salesman problem; Moret et al. (2001b) use approximate TSP solvers, Sankoff et al. (2000) a heuristic median solver. In `GRAPPA`, both Caprara's and Siepel's median solvers are implemented (Caprara, 2001; Siepel and Moret, 2001).

Large phylogeny. In `BPAnalysis` and `GRAPPA`, the best phylogenetic tree is found by exhaustive search – such approaches are only possible for a very limited number of species (around 13, since there are $(2n-5)!!$ unrooted trees on n labeled leaves). During the search, it is important to be able to estimate the best score of a tree to discard most of the topologies that do not look promising (Moret and Tang (2005) report 500-fold speedup obtained by improving the tree lower bound of `BPAnalysis`).

If σ is a circular permutation of the leaves of tree T under some planar embedding of T, then $C_\sigma(T) = {}^1/_2 \sum d(\pi_{\sigma_i}, \pi_{\sigma_{i+1}})$ is the *circular lower bound* of T (Sankoff et al., 1996; Moret et al., 2002a). Maximum over all circular orderings can be computed in $O(n^2)$ (Bachrach et al., 2005), however, heuristics are preferred in the implemented software.

A better lower bound can be computed using linear programming (Bachrach et al., 2005; Tang, 2005).

Sequential addition. Since there are too many possible tree phylogenies, Bourque and Pevzner (2002) proposed to build a single phylogenetic tree directly. The idea is to greedily add new genomes to an already partially built tree. To add a new genome π to the tree, we need to find a suitable edge $\{u, v\}$ with genomes π_u, π_v and replace it by a 3-star. Genome π then becomes a new leaf and the genome at the middle vertex is a median $M_{u,v} \in \mathfrak{M}(\pi, \pi_u, \pi_v)$. The cost of adding π at $\{u, v\}$ is

$\Delta(u,v) = S(M_{u,v}) - d(\pi_u, \pi_v)$ and we always search for an edge $\{u,v\}$ which minimally affects the total score, i.e., one with the minimum $\Delta(u,v)$. This method is implemented in the `MGR` software.

One advantage of this approach is that we do not have to start from scratch, but we can add new genomes to an already built tree (possibly constructed by a different method). On the other hand, a drawback of this method is that the reconstructed phylogenetic tree depends on the ordering of the input genomes and fixing one (out of many) medians permanently affects the score of the tree and all the subsequent choices. Therefore, Bernt et al. (2007) propose to abandon the definitive choices. Instead, in each step, they choose one of the remaining genomes, a splitting edge, generate the whole set $\mathfrak{M}(\pi, \pi_u, \pi_v)$, and try out several optimal medians. These modifications are implemented in `amGRP`.

5.4 Maximum Likelihood Methods

Another approach to reconstructing phylogenies is to design a probabilistic model of evolution and then search the most likely parameters. If θ are parameters of our model and D are observed data, *likelihood* of parameters θ is $L(\theta) = L(\theta \mid D) = \Pr(D \mid \theta)$. Note that by Bayes' theorem, $\Pr(\theta \mid D) = \text{const} \cdot \Pr(\theta) \cdot L(\theta)$, so if the parameters θ have uniform prior distribution, the posterior probability $\Pr(\theta \mid D)$ is proportional to the likelihood of θ.

Sequence data. We assume that sequences evolve by substitutions, which are usually modeled as a continuous random walk on the bases `A`, `C`, `G`, `T`, and we derive the probability $\Pr(y \mid x, t)$ of sequence x evolving into y by point mutations over time t. Several substitution models were proposed and studied, see Jukes and Cantor (1969); Kimura (1980); Felsenstein (1981); Hasegawa et al. (1985); Tamura (1992); Tamura and Nei (1993).

Again, by bottom-up dynamic programming (Felsenstein, 1981), it is possible to compute probability $\Pr(\vec{X} \mid T, \vec{t})$ of input data $\vec{X}$ (sequences of extant species) given a phylogenetic tree T and times on its branches $\vec{t}$. This is the likelihood of the parameters T and $\vec{t}$ ($L(T, \vec{t}) = \Pr(\vec{X} \mid T, \vec{t})$). The *maximum likelihood* methods search for T and $\vec{t}$ which maximize the likelihood function.

Given a phylogenetic tree T, the branch lengths which maximize the likelihood can be found by expectation-maximization, hill-climbing, or other standard optimization algorithms. The computational complexity of this problem is unknown.

Chor and Tuller (2006) proved that maximum likelihood reconstruction of phylogenetic trees is NP-hard. Moreover, even approximating the logarithm of the maximum likelihood within a factor of 1.00175 is hard (Chor and Tuller, 2005). On the other hand, Elias and Tuller (2007) show an FPT and a 2-approximation algorithms solving this problem.

In practice, to maximize over all trees, the best tree is searched exhaustively or heuristically by topological changes of an initial tree (see Nei and Kumar (2000); Guindon and Gascuel (2003); Stamatakis et al. (2005)). Such methods are implemented in many software packages which are available for phylogeny reconstruction, such as `PAUP*` (Swofford, 2003), `MacClade` (Maddison and Maddison, 2000),

`Mesquite` (Maddison and Maddison, 2004), `Phylip` (Felsenstein, 2002), `MEGA5` (Tamura et al., 2011), `PhyML` (Guindon et al., 2010), or `RAxML` (Stamatakis, 2006).

Another possibility is to sample trees and branch lengths from the posterior probability $\Pr(T, \vec{t} \mid \vec{X})$. A notable example is `MrBayes` software (Ronquist and Huelsenbeck, 2003), which is based on the Markov Chain Monte Carlo algorithm (MCMC). The main idea is to perform a random walk on all trees and branch lengths. In each step, a new tree and new branch lengths are proposed, and these can be either accepted or rejected by the Metropolis-Hastings method. By rejecting proposals with a suitable probability, we can ensure that the stationary distribution (which is attained in the limit) is $\Pr(T, \vec{t} \mid \vec{X})$. In other words, after making sufficiently many steps, the probability of the random walk being in state $(T, \vec{t})$ is approximately $\Pr(T, \vec{t} \mid \vec{X})$. Sampling of trees and branch lengths is done by sampling the states of the random walk.

Gene-order data. For gene-order data, an MCMC algorithm was proposed by Larget et al. (2005) (implemented in `BADGER`). In their model of evolution, all unrooted trees are equally likely. The branch lengths are selected independently from a gamma distribution and given length ℓ, a Poisson number of reversals with mean ℓ are realized. Only reversal operations are allowed and all reversals have the same probability.

One advantage of probabilistic inference is that we also obtain probabilities for individual edges, so we know, which edges are more and which are less certain. Another advantage pointed out by Larget et al. (2005) is, that the most parsimonious solutions do not necessarily occur with the highest probability. Consider these two examples:

$$\pi_1 = (\overrightarrow{8}, \overrightarrow{3}, \overrightarrow{7}, \overrightarrow{1}, \overleftarrow{5}, \overleftarrow{4}, \overleftarrow{6}, \overrightarrow{2}) \text{ and } \pi_2 = (\overrightarrow{2}, \overrightarrow{3}, \overrightarrow{4}, \overrightarrow{5}, \overrightarrow{6}, \overrightarrow{8}, \overrightarrow{1}, \overrightarrow{7}).$$

Since $rev(\pi_1) = 4$ and $rev(\pi_2) = 5$, we may be inclined to say that, starting from the identity permutation, it is more probable to generate π_1 than π_2 by random reversals. Actually, the contrary is true. The reason is that π_2 can be obtained from the identity in much more different ways. There is just one sorting scenario of length 4 and only 8 sorting scenarios of length 5 for π_1, while π_2 has 200 sorting scenarios of length 5. While π_2 has 2 668 and 147 282 scenarios of length 6 and 7, π_1 has only 791 and 9 918.

Part II

Contributions

Chapter 6

Restricted DCJ model

6.1 Introduction

In this chapter, we study classical problems of genome rearrangement – *sorting*, *halving*, and *median* problems – in a restricted double cut and join model.

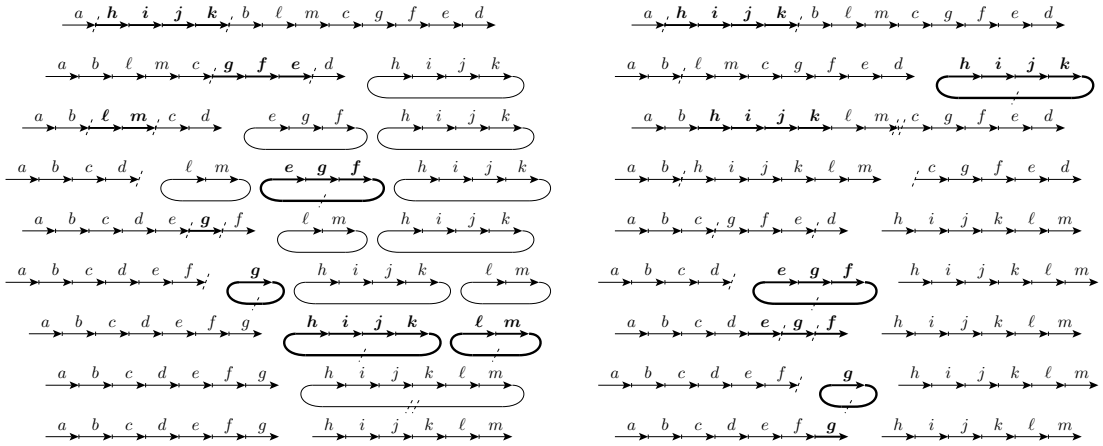
The DCJ model, described in Section 2.2, was introduced by Yancopoulos et al. (2005) to model most of the large-scale mutation events, such as reversals, translocations, fusions, and fissions in a unified way. Transpositions and block interchanges can be simulated in this model by two operations: an appropriate segment of a chromosome is extracted, creating a temporary circular chromosome, and then reinserted at the proper place in the next step.

The generalization by Bergeron et al. (2006b) further simplified the model by allowing mixed genomes containing both linear and circular chromosomes. They proposed a simple linear-time algorithm for DCJ sorting which finds a sequence of DCJ operations without any explicit mention of the underlying operations (reversals, translocations, block interchanges, etc.). This algorithm does not keep track of the chromosomes and in consequence, many circular chromosomes may coexist at intermediate stages of the sorting process (see Fig. 6.1(a)).

However, when working with multilinear genomes (e.g., genomes of eukaryotic organisms), such sorting sequences are not biologically plausible. The intermediate circular chromosomes are thus considered an artifact of the DCJ model.

In this chapter, we revisit the model from the original study of Yancopoulos et al. (2005), restricting the DCJ model to genomes with multiple linear chromosomes. In particular, we require that each excision of a circular chromosome is always immediately followed by its reincorporation. This is equivalent to sorting by reversals, translocations, fusions and fissions, and block interchanges with weight 2 (see Fig. 6.1(b)). Interestingly, this restriction does not change the distance and Yancopoulos et al. (2005) gave a quadratic algorithm for finding an optimal restricted scenario.

We borrow techniques from other studies on sorting by reversals and block interchanges (Christie,



(a) An optimal DCJ sorting scenario.

(b) Restricted DCJ sorting scenario.

Figure 6.1: Two optimal DCJ sorting scenarios transforming the first genome into the last one in eight operations. In the unrestricted model, the scenario on the left is a possible result of sorting. Even though the number of operations is minimal, the scenario is not biologically plausible: We started and finished with linear chromosomes, but the intermediate genomes contain many circular chromosomes. In contrast, the scenario on the right is restricted: each circular excision is immediately followed by its reincorporation. Thus, we can, for instance, treat the first circular excision plus reintegration as a transposition of the block (h, i, j, k) between markers b and ℓ.

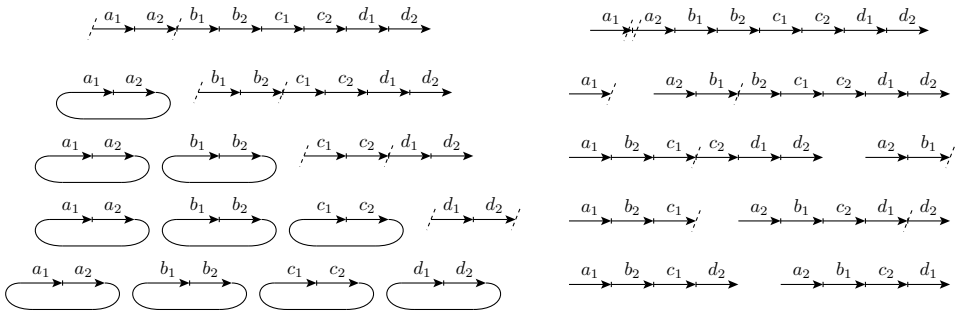
1996; Feng and Zhu, 2007; Swenson et al., 2010) and propose a new algorithm that sorts multichromosomal linear genomes in the restricted DCJ model in $O(n \log n)$ time. This improves the original result by Yancopoulos et al. (2005).

Furthermore, we present a new result on the halving problem. If no restriction on the linearity of chromosomes is imposed, and no guarantee concerning circular reintegration is required, we can use linear-time algorithms proposed by Warren and Sankoff (2009b) and Mixtacki (2008) to reconstruct the ancestor. However, given a multilinear genome, these algorithms may predict circular chromosomes in the ancestral genome. In the worst case, these algorithms may even produce $\Omega(n)$ circular chromosomes given a single linear chromosome of length n (see Fig. 6.2). Again, this is not biologically plausible, when organisms with linear genomes are considered.

The restricted halving problem has not been studied previously and is stated as open in Tannier et al. (2009). In Section 6.3, we propose a new algorithm that reconstructs a *multichromosomal linear* perfectly duplicated ancestor in linear time. One particular restricted halving scenario can be found in $O(n \log n)$ time.

Finally, in Section 6.4, we show that the median problem is NP-hard in the restricted DCJ model, as conjectured by Tannier et al. (2009).

This chapter is based on joint work with Robert Warren, Marília Braga, and Jens Stoye during my research visit at Bielefeld University in Germany. The results were presented at the RECOMB-CG 2010



(a) An optimal perfectly duplicated ancestor in the DCJ model.

(b) An optimal perfectly duplicated ancestor in the restricted DCJ model.

Figure 6.2: Two different halving scenarios of the same genome at the top. The halving distance is the same in both cases, but the algorithm of Mixtacki (2008) predicts that the ancestor of our linear genome had four circular chromosomes, which is biologically implausible.

conference (Kováč et al., 2010) and published in *Journal of Computational Biology* (Kováč et al., 2011b).

6.2 Restricted DCJ Sorting

Bergeron et al. (2006b) gave a linear-time algorithm for DCJ sorting, disregarding the constraint of immediate reincorporation of circular chromosomes. The solution can be easily adapted to a quadratic-time algorithm for the restricted version: after each step, check whether a circular chromosome was created and if so, find the appropriate DCJ operation acting on adjacencies in the circular and the original linear chromosome that reintegrates the circular chromosome. It is not clear how to do this efficiently (say, in polylogarithmic time).

Yancopoulos et al. (2005) proposed to transform genome π into γ by restricted sorting in four stages:

0. Add caps at the ends of linear chromosomes.

1. By translocations, fusions, and fissions, transform π into π', where π' is a genome with chromosomes that have the same marker contents as chromosomes in γ.

2. Perform oriented reversals to transform π' into π'', where orientation of all markers in genome π'' matches the orientation of markers in γ.

3. Finally, use block interchanges to transform π'' into γ.

Stages 2 and 3 can be implemented in $O(n \log n)$ time using the data structure described in Section 6.2.2 (Swenson et al., 2010; Feng and Zhu, 2007). Thus, a *unichromosomal* restricted DCJ sorting can be solved easily in $O(n \log n)$ time. However, it is not clear, how to implement stage 1 efficiently. (Actually, we hypothesize that it is not possible in polylogarithmic time.)

In the rest of this section, we first introduce our new algorithm for restricted DCJ sorting. In Section 6.2.2, we describe the data structure necessary to implement this algorithm efficiently and in Section 6.2.3, we comment on perfect restricted sorting scenarios.

6.2.1 Algorithm

Our algorithm is based on the following observation:

Observation 1. *Let g, h be two markers that are adjacent in γ, but not in π. If g and h are on different chromosomes in π, there is a translocation that puts them together. If g and h are on the same chromosome and have a different orientation, there is a reversal that puts them together. These operations are optimal in the DCJ model. Transposition and block interchange take two DCJ operations. These operations are optimal if they create two new non-telomeric common adjacencies and destroy none.*

This is simply because, even more generally, k operations, that create k new non-telomeric adjacencies and destroy none, create k new cycles in the adjacency graph, and thus decrease the distance by k.

Theorem 10. *A restricted optimal DCJ sorting scenario transforming multilinear genome π into multilinear genome γ can be found in $O(n \log n)$ time.*

Proof. The ends of linear chromosomes, telomeres, produce some difficulties and nasty special cases. Capping is an elegant technique to deal with them: the idea is to adjoin new markers (called *caps*) to the ends of chromosomes in such a way that we do not change the distance but we do not have to worry about telomeres any more.

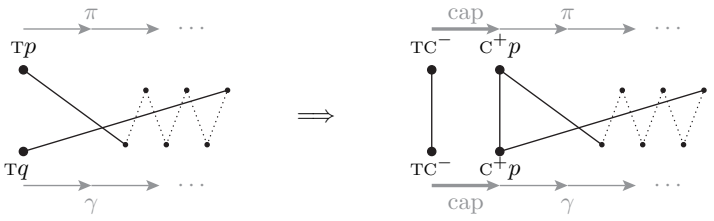
We find all the paths in the adjacency graph $AG(\pi, \gamma)$. Paths of odd length have one end in π and one in γ – simply adjoin a new marker (properly oriented) to the two telomeres (see Fig. 6.3(a)). This increases the number of markers by one, but instead of an odd path, we have a cycle and a 1-path, so the distance does not change. For paths starting and ending in π, add two new markers to the ends of π and add a new chromosome consisting of just these two markers (properly oriented) to γ (see Fig. 6.3(b)). The case with a path starting and ending in γ is symmetric. The number of markers increases by 2, but instead of an even path, we have a cycle and two 1-paths, so the distance does not change. Capping of all chromosomes can be done in linear time.

Without loss of generality, we may assume that the markers in chromosomes of γ are consecutive numbers $(\overrightarrow{k_0}, \ldots, \overrightarrow{k_1 - 1}), (\overrightarrow{k_1}, \ldots, \overrightarrow{k_2 - 1}), \ \ldots \ , (\overrightarrow{k_{s-1}}, \ldots, \overrightarrow{k_s - 1})$ where $0 = k_0 < k_1 < k_2 < \cdots < k_s = n$ (otherwise renumber the markers).

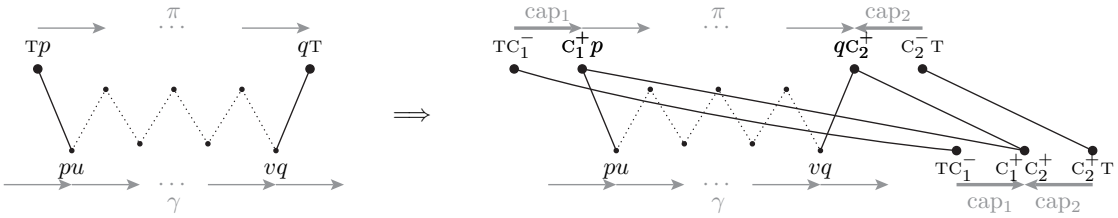
We will be transforming π into γ gradually "from left to right": once we have transformed the beginning of a chromosome in π to $\overrightarrow{k_i}, \overrightarrow{k_i + 1}, \ldots, \overrightarrow{j}$, we extend it by moving $j + 1$ next to $\overrightarrow{j}$.

There are several cases we need to consider:

1. If $\overrightarrow{j + 1}$ is already next to $\overrightarrow{j}$, we are done.



(a) Capping odd paths: an odd path in $AG(\pi, \gamma)$ is transformed into a 1-path and a cycle.



(b) Capping even paths: an even path in $AG(\pi, \gamma)$ is transformed into two 1-paths and a cycle.

Figure 6.3: Capping the chromosomes: We add new markers (caps) at the ends of chromosomes. Note that the DCJ distance preserved and after the transformation, the two genomes have all the telomeres in common and all the paths in their adjacency graph are 1-paths.

2. If $j+1$ is on a different chromosome than $\overrightarrow{j}$, we can always use a translocation. In the rest of the proof, we assume that $j+1$ is on the same chromosome, located to the right of $\overrightarrow{j}$.
3. If $\overrightarrow{j}$ and $\overleftarrow{j+1}$ have different orientation, we can use a reversal.

Otherwise, following Christie (1996), find the marker m with the highest number between $\overrightarrow{j}$ and $\overrightarrow{j+1}$ and find $m+1$.

4. If $m+1$ is on a different chromosome, we can use a translocation to move it next to m; this operation also moves $\overrightarrow{j+1}$ to another chromosome, so we can use another translocation to move it next to $\overrightarrow{j}$.

Otherwise the situation is $\overrightarrow{j}, \ldots, m, \ldots, \overrightarrow{j+1}, \ldots, m+1$ (since m is the highest number between $\overrightarrow{j}$ and $\overrightarrow{j+1}$ and the part of the chromosome to the left of $\overrightarrow{j}$ is already sorted, $m+1$ must be located to the right of $\overrightarrow{j+1}$).

5. If m and $m+1$ have different orientation, we can use a reversal to move $m+1$ next to m; this will also change the orientation of $\overrightarrow{j+1}$, so in the next step, we can use another reversal to move $\overleftarrow{j+1}$ next to $\overrightarrow{j}$.
6. Finally, if m and $m+1$ have the same orientation, we interchange blocks

$$\overrightarrow{j}, [\ldots, \overrightarrow{m}], \ldots, [\overrightarrow{j+1}, \ldots], \overrightarrow{m+1} \quad \rightsquigarrow \quad \overrightarrow{j}, \overrightarrow{j+1}, \ldots, \overrightarrow{m}, \overrightarrow{m+1},$$

if both $\overrightarrow{m}$ and $\overrightarrow{m+1}$ have positive direction and

$$\overrightarrow{j}, [\ldots], \overleftarrow{m}, \ldots, [\overrightarrow{j+1}, \ldots, \overleftarrow{m+1}] \quad \rightsquigarrow \quad \overrightarrow{j}, \overrightarrow{j+1}, \ldots, \overleftarrow{m+1}, \overleftarrow{m},$$

if $\overleftarrow{m}$ and $\overleftarrow{m+1}$ have both negative direction. By two operations we move $\overrightarrow{j+1}$ to $\overrightarrow{j}$ and $\overrightarrow{m}$ to $\overrightarrow{m+1}$.

Every step can be implemented in $O(\log n)$ time using the data structure described in the next section. □

6.2.2 Data Structure for Handling Permutations

Our sorting algorithm uses a data structure for handling permutations by Kaplan and Verbin (2005). It can be traced back to Chrobak et al. (1990), where it was used to improve heuristics for the traveling salesman problem. It supports the following three operations in logarithmic time:

1. find the i^{th} marker in a linear chromosome,
2. return the position of marker g, and
3. perform a reversal operation.

Linear chromosomes can be represented by a balanced tree supporting operations split and merge (e.g. red-black tree or splay tree). The order is the same as the left-to-right order of markers on the chromosome. In each node of the tree, we store one marker, its orientation, number of descendants, and a reverse flag. A reverse flag being "on" signifies that the whole subtree is reversed. The reverse flag of node v can be cleared ("pushed down") by changing v's orientation, swapping its children and flipping their reverse flags.

Reversing a segment from i to j can be implemented as follows:

1. Find the i^{th} and j^{th} marker (using the information about sizes of subtrees and reverse flags).
2. Split the tree into three parts: T_1 with markers located before i, T_3 with markers located after j, and T_2 with the segment from i to j.
3. Flip the reverse flag in the root of T_2, and
4. Merge T_1, T_2 and T_3.

We store a lookup table with a pointer to the corresponding node of a tree for every marker. In this way, we can find the position of any marker in logarithmic time.

This data structure can be easily extended to support multiple linear chromosomes and the following operations required in our sorting algorithm:

1. Find the chromosome that contains a given marker.

2. Perform a DCJ operation.

3. Given an interval from i to j, find the marker with the highest number on the chromosome between i and j.

To support multilinear genomes, we simply concatenate the chromosomes with a delimiter between each pair, and in each node, we store the number of delimiters in its subtree. This way, given a marker g, we can determine its chromosome simply by counting the number of delimiters before g.

To support different rearrangement operations, we can express them as a sequence of reversals. For example, block interchange can be mimicked by four reversals; if we add sufficiently many delimiters at the end of the sequence (representing empty chromosomes), we can also mimic fusions and fissions.

To support the last query, we store the highest number in the subtree in each node.

6.2.3 Perfect DCJ scenarios

Bérard et al. (2009) studied the problem of finding a scenario transforming genome π into γ that does not break a given set of common intervals. An *interval* in genome π is a set of markers such that the subgraph of G_π induced by their extremities is connected. Intervals of π have zero or two borders – adjacencies such that one extremity is inside and one outside. Let I be any set of markers with zero or two borders. A DCJ operation *preserves* I, if I still has zero or two borders in the resulting genome. Bérard et al. (2009) showed that for nested sets of common intervals (when the intervals do not overlap), the shortest scenario can be found in polynomial time and for weakly separable sets, the problem is NP-hard but fixed parameter tractable.

Since their algorithms use algorithms for DCJ distance and sorting as a black box, one can use them in conjunction with our algorithm to get perfect restricted DCJ scenarios.

6.3 Restricted DCJ Halving

The simple approach that works for sorting – perform a DCJ operation, test whether a circular chromosome was created and reincorporate it – does not work for halving: In some cases, the circular chromosome cannot be reincorporated. For example, take chromosome $(a_1, a_2, \ldots, d_1, d_2)$ from Fig. 6.2(a). After excision of circular chromosome $[a_1, a_2]$, it is not possible to reincorporate it and the algorithm of Mixtacki (2008) ends with four (perfectly duplicated) circular chromosomes. On the other hand, by fissions and translocations, we can get the restricted scenario as shown in Fig. 6.2(b).

We describe a new algorithm finding a multilinear doubled ancestor in linear time. A particular restricted halving scenario can then be found in $O(n \log n)$ time by sorting. We also prove that the halving distance is always the same in the restricted and the unrestricted case, and can be computed in linear time from the natural graph (Mixtacki, 2008).

Our new algorithm works in three steps:

1. First, we use the halving algorithm by Mixtacki (see Section 4.3) that computes a doubled genome in linear time.

2. In general, the result may contain circular chromosomes. Thus, in the next step, we transform it into a multilinear doubled genome. We prove that this can be done in linear time while preserving optimality of the result.

3. Finally, if a particular halving scenario is needed, we can use the sorting algorithm from Section 6.2 to obtain a scenario transforming the given genome into a doubled genome.

Theorem 11. *Given a duplicated genome π and a doubled genome θ – a result of halving π, we can find a* multilinear *doubled genome θ' within the same DCJ distance from π in linear time.*

Proof. For convenience, assume that π and θ are properly capped. The capping procedure is similar to the one described in Section 6.2. However, we will treat the corresponding caps at the beginning of two paralogous chromosomes in θ as paralogs and for each added chromosome consisting of two caps, we add a paralogous copy to both π and θ. This way we ensure that

1. both π and θ contain exactly two copies of each marker,

2. θ is a doubled genome (in particular, every linear chromosome has an exact second copy in θ),

3. the distance between π and θ is preserved, and

4. for each (linear) chromosome in π, there is a linear chromosome in θ beginning with the same marker.

The main idea of our "linearization" algorithm is to search for adjacencies $\{x, y\}$ in π such that in θ, x belongs to a *linear* chromosome, while y belongs to a *circular* chromosome. We will call such an adjacency $\{x, y\}$ an *integrating* adjacency, since if $\{x, p\}$ and $\{y, q\}$ are the adjacencies in θ, and we replace

$$\{x, p\}, \{y, q\} \quad \text{with} \quad \{x, y\}, \{p, q\} \text{ and} \tag{6.1}$$

$$\{\overline{x}, \overline{p}\}, \{\overline{y}, \overline{q}\} \quad \text{with} \quad \{\overline{x}, \overline{y}\}, \{\overline{p}, \overline{q}\}, \tag{6.2}$$

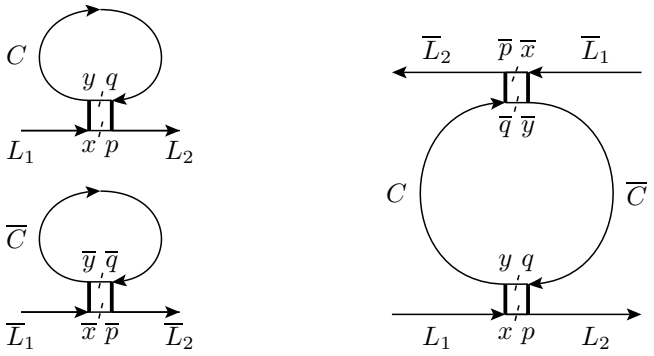
we incorporate the circular chromosome (and its copy) into the linear chromosome (and its copy). Our algorithm searches for integrating adjacencies in π and incorporates circular chromosomes.

It remains to be proven that:

1. Replacing the adjacencies results in a doubled genome (i.e., the result is still perfectly duplicated), with fewer circular chromosomes.

2. Replacing the adjacencies preserves the DCJ distance $dcj(\pi, \theta)$.

3. If there is no integrating adjacency in π, there are *no* circular chromosomes in θ.



(a) The resulting chromosomes are (L_1, C, L_2) and $(\overline{L}_1, \overline{C}, \overline{L}_2)$.

(b) The resulting chromosomes are $(L_1, C, \overline{L}_2)$ and $(\overline{L}_1, \overline{C}, L_2)$.

Figure 6.4: We transform a doubled genome by incorporating circular chromosomes. We cut adjacencies $\{x, p\}$, $\{y, q\}$ and replace them by $\{x, y\}$, $\{p, q\}$ (bold). We do the same with the paralogous copies. There are two cases, depending on whether $\{y, q\}$ and $\{\overline{y}, \overline{q}\}$ are in two paralogous chromosomes (left), or in one doubled chromosome (right). Note that in both cases, the resulting genome is a doubled genome with fewer circular chromosomes.

4. The algorithm can be implemented in linear time.

Claim 1: Figure 6.4 illustrates the proof of the first claim.

Claim 2: To see the second claim, let θ be the genome before, and θ' the genome after the two replacements. Note that the replacements (1) and (2) are actually two DCJ operations, so the difference between $dcj(\pi, \theta')$ and $dcj(\pi, \theta)$ cannot be more than two. Furthermore, note that the first operation decreases the distance between π and θ, so $dcj(\pi, \theta') \leq dcj(\pi, \theta)$. However, since θ is optimal, we also have $dcj(\pi, \theta) \leq dcj(\pi, \theta')$.

Claim 3: Let m be a marker belonging to a circular chromosome in θ and let L be the chromosome in π containing m. We can write L as $L = (m_1, m_2, m_3, \ldots, m, \ldots)$. Thanks to capping, m_1 belongs to a linear chromosome in θ. Let m_{k+1} be the first marker of L belonging to a circular chromosome in θ, then there is an adjacency between m_k and m_{k+1} in π such that the first extremity belongs to a linear chromosome and the second one belongs to a circular chromosome in θ. In other words, if θ has a circular chromosome, there is always an integrating adjacency in π. This proves our third claim.

Claim 4: We implement the algorithm as follows: First, we traverse all chromosomes in θ and populate a lookup table that stores the type of chromosome (linear or circular) for each extremity. Let ℓ be the list of all extremities in linear chromosomes of θ (this list will grow as we incorporate circular chromosomes). For each extremity x in ℓ, let $\{x, y\}$ be an adjacency in π and let C be the chromosome in θ containing y. If $\{x, y\}$ is integrating, i.e., if C is a circular chromosome, we set the chromosome type of each extremity of C to linear in the lookup table, we append these extremities to the list ℓ and incorporate the circular chromosome(s) by replacing adjacencies as in (6.1) and (6.2).

The algorithm obviously runs in linear time since for each extremity, we determine only once whether

it belongs to a linear or a circular chromosome. We change this information at most once (when a circular chromosome is incorporated in a linear one), and once the extremity belongs to a linear chromosome, we check at most once whether the extremity adjacent in π belongs to a circular chromosome. □

Corollary 2. *The halving distance is the same in the DCJ and in the restricted DCJ model. The distance and one multilinear doubled genome can be computed in linear time. A restricted halving scenario can be computed in time* $O(n \log n)$.

6.4 Restricted DCJ Median

In sorting and halving problems, the reincorporation restriction did not change the distance, and we were just searching for restricted scenarios that better explain the evolutionary history. This is not the case for the median problem: Consider three linear genomes $(1, 2, 3)$, $(2, 1, 3)$, and $(2, 3, 1)$. Their median in the unrestricted case consists of a linear chromosome $(2, 3)$ and a circular chromosome [1]. Since we can obtain any of the linear genomes by a single operation (reincorporation of chromosome [1]), the median score is 3. This score, however, cannot be achieved in the restricted model. Generalizing this pattern, we can get genomes of length $3n$, with unrestricted median score $3n$, and restricted median score $4n$.

Caprara (2003) proved that finding a median in a DCJ model restricted to unichromosomal linear genomes[1] is NP-hard. The median problem in the restricted DCJ model was not studied; it was conjectured NP-hard, but stated as open in Tannier et al. (2009). We show that the NP-hardness follows from the result of Caprara (2003):

Theorem 12. *The median problem is NP-hard in the restricted DCJ model.*

Proof. We show that if we could solve the *multichromosomal* linear DCJ median efficiently, we could also solve the *unichromosomal* linear DCJ median efficiently. Let π_1, π_2, π_3 be three (unichromosomal) linear genomes and let M be their multilinear median. We show that we can join telomeres of M and fuse chromosomes into a single chromosome M' while preserving the median score.

Consider two linear chromosomes in M. Each of them has two telomeres, so *we can join them in four different ways.* What happens to the distance $dcj(M, \pi_i)$, if we join two telomeres? Recall that the DCJ distance between two genomes is calculated as $n - (c + p_o/2)$, where c is the number of cycles, and p_o is the number of odd length paths in their adjacency graph.

1. If the telomeres belong to two different paths of even length, the distance remains unchanged.
2. If the telomeres belong to the same path of even length, by joining them, we create a cycle and the distance decreases.
3. Finally, if the telomeres belong to two (different) odd length paths, by joining the telomeres, we get a path of even length and the distance increases. This is the *bad* case.

[1] Caprara called it the *cycle median problem* – the DCJ model did not exist yet

However, since each of the input genomes π_1, π_2, π_3 is linear, it has only two telomeres and the adjacency graph $AG(\pi_i, M)$ may contain at most two paths of odd length. That is, if the bad case occurs, all of the other three possibilities of joining the telomeres are good (do not change the distance, or can even decrease it). Since there are four ways of joining the telomeres, but for each genome π_1, π_2, π_3, at most one way corresponds to the bad case, we can always fuse the linear chromosomes without increasing the total distance. The solution to the multichromosomal restricted DCJ median can thus be transformed into a solution to the unichromosomal restricted DCJ median. □

6.5 Conclusion

In this chapter, we have revisited the restricted DCJ model for multichromosomal linear genomes, where a temporary circular chromosome is immediately reincorporated after its excision. We improved on the quadratic-time algorithm by Yancopoulos et al. (2005), and proposed an algorithm that runs in $O(n \log n)$ time. Furthermore, we have solved an open problem from Tannier et al. (2009) by giving an algorithm for the restricted halving problem. The algorithm shows that the halving distance for the restricted version is the same as the distance for the unrestricted version, and given a multilinear duplicated genome, an optimal *multilinear* perfectly duplicated genome can always be found in linear time. Finally, we confirmed the conjecture from Tannier et al. (2009), and proved that the median problem is NP-hard in the restricted DCJ model.

Chapter 7

PIVO

7.1 Introduction

This chapter is based on joint work with Broňa Brejová and Tomáš Vinař. It was motivated by our collaboration with Matúš Valach, Ľubomír Tomáška, and Jozef Nosek from the Faculty of Natural Sciences of Comenius University and other researchers who studied mitochondrial genomes of yeasts from the 'CTG clade' of *Hemiascomycetes*. This clade (see Fig. 7.1) is very interesting from the biological point of view, since it contains closely related species with great genome architecture diversity. While *C. subhashii, C. parapsilosis,* and *C. orthopsilosis* are linear, *C. frijolesensis* has two linear chromosomes, and the rest of the species have circular chromosomes.

The ultimate goal is to understand this variety and the mechanisms leading to it; to describe molecular processes affecting the genome architecture; to explain the origin and mechanisms for the maintenance of telomeres. We have already mentioned that most of the prokaryotic genomes are circular, while eukaryotic genomes have multiple linear chromosomes. Evolution and the biological role of these linear chromosomes is one of the big open problems in biology. To shed some light on what was happening during the evolution of these species, we were interested in reconstructing their rearrangement history. Here, we present our work on developing a new method and implementing a new software for such reconstructions.

The motivation behind this project was twofold: The first, more pragmatic reason was that there was simply no software available which could accommodate such variety of genome architectures (single vs. multiple chromosomes, linear vs. circular chromosomes; most of the existing software accepts only linear or multilinear genomes). In our software called `PIVO` (Phylogeny by IteratiVe Optimization), we have implemented the DCJ, restricted DCJ, and reversal models for measuring rearrangement distance and, of course, our method can be easily applied to other rearrangement distance measures.

In Section 5.3, we have reviewed methods for reconstructing evolutionary history – the so called small phylogeny problem. By far the most popular approach is the Steinerization method and it seems

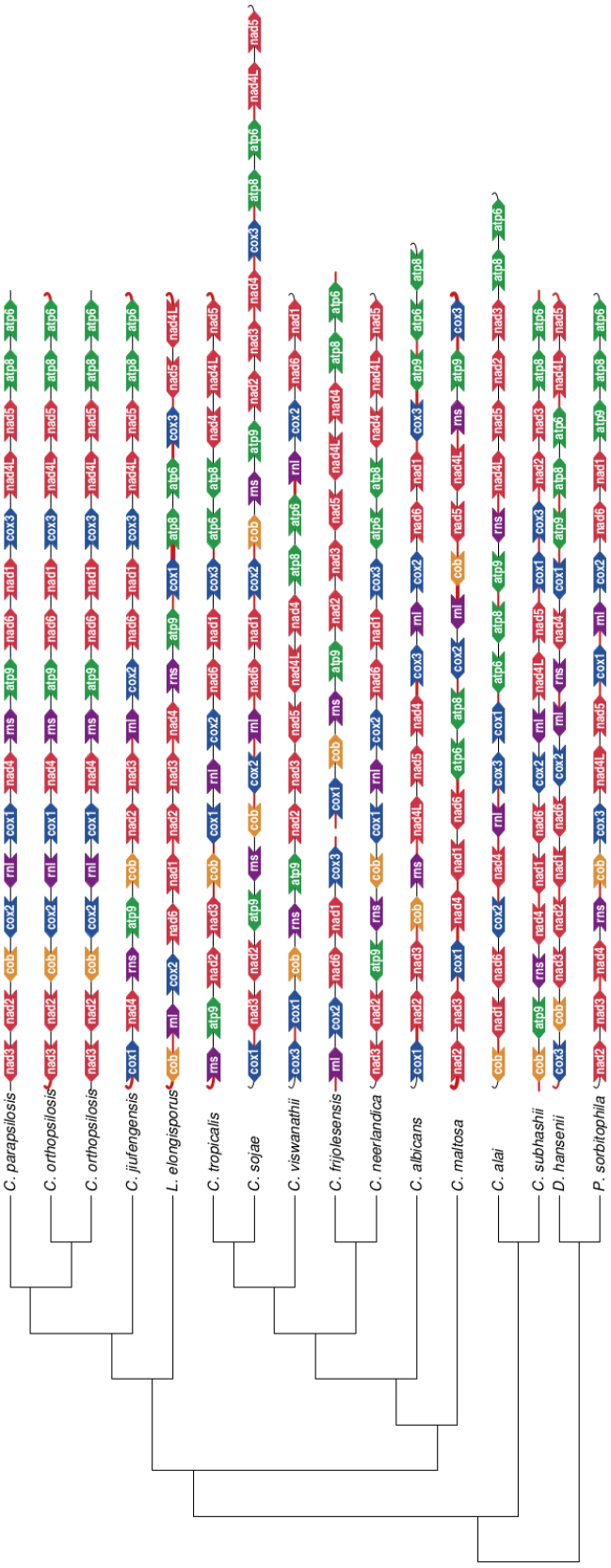


Figure 7.1: Phylogenetic tree of 16 species of the *Hemiascomycetes'* clade and their mitochondrial genomes. Due to space constraints, only selected markers are shown. Rounded ends of chromosomes designate circular chromosomes.

that the trend in current research is to come up with fast heuristics which enable analyses of high-resolution whole-genome data with thousands of markers. In contrast, we were interested in analysing small mitochondrial genomes, and getting more precise results even at the cost of being slower than the other solutions. Thus our second, more ambitious goal was to develop a better method for reconstructing evolutionary history in this setting.

We describe our new method based on proposing candidate ancestral genomes and choosing the best combination in the next section. Then, in Section 7.3, we show experimental results and compare `PIVO` with other existing solutions. We demonstrate the accuracy of our program on the well-studied dataset of *Campanulaceae* chloroplast genomes (Cosner et al., 2000), and apply it to the reconstruction of rearrangement histories of newly sequenced mitochondrial genomes of pathogenic yeasts from *Hemiascomycetes* clade (Valach et al., 2011).

This chapter is based on the paper presented at the WABI 2011 conference (Kováč et al., 2011a). Our results on the yeast genomes also made a contribution to the biological paper by Valach et al. (2011) published in the *Nucleic Acids Research* journal.

7.2 Methods

In this section, we introduce a new general approach to the small phylogeny problem based on iterative local optimization. The basic idea is that in each step, we propose multiple candidates for ancestral genomes in each internal node of the tree and choose the most parsimonious combination of the candidates by dynamic programming. We will formulate the method in general terms for any rearrangement distance measure d that can be efficiently computed and for any strategy for proposing candidate ancestral genomes. Specific strategies will be discussed in Section 7.2.2

Recall from Section 5.3 that given a phylogenetic tree $T = (V, E)$ with the set of leaves L and genomes of extant species $\pi : L \to \mathcal{G}$, our goal is to find an evolutionary history $H : V \to \mathcal{G}$ which minimizes the score

$$S(H) = \sum_{\{u,v\} \in E} d(H(u), H(v)).$$

We start with some history H_0. For a particular history H and each internal vertex v, we propose a set of candidates $\text{cand}(H, v)$. We define a neighbourhood of history H as the set of all possible combinations of candidate genomes $\mathfrak{N}(H) = \{H' \mid \forall v \in V : H'(v) \in \text{cand}(H, v)\}$. We then find the best history in the neighbourhood $\mathfrak{N}(H)$ by a dynamic programming algorithm. If the new history is better than the previous one, we take it and repeat the iteration. Otherwise, we have found a local minimum and the algorithm terminates. We repeat the local optimization several times starting from different histories H_0. Algorithm 2 summarizes the local optimization method.

Example #1: For each internal vertex v, the set of candidates $\text{cand}(H, v)$ can be the set of all the genomes within the distance 1 from $H(v)$. The neighbourhood of H is then the set of all histories, we can obtain from H by performing at most one operation to each ancestral genome. Note that the size

Algorithm 2: Iterative local optimization

Data: phylogenetic tree T and initial evolutionary history H

Result: locally optimal evolutionary history

```
s' ← S(H), s ← ∞ ;
while s' < s do
    cand ← generate lists of candidates (neighbourhood of H);
    H ← best(cand);
    s ← s', s' ← S(H);
end
return H
```

of $\mathfrak{N}(H)$ is exponential in the number of internal vertices, but as we will see later, we will never have to enumerate the entire neighbourhood.

Example #2: The Steinerization approach mentioned in Section 5.3 is a special case of our method: Here, $H(v)$ is the only candidate for all vertices except for one vertex w with neighbours a, b, c, for which there are two candidates: the current genome $H(w)$ and one possible median of the genomes in the neighbouring vertices:

$$\text{cand}(H, v) = \begin{cases} \{\, H(v) \,\} & \text{for } v \neq w \\ \{\, H(w), M \,\} & \text{where } M \in \mathfrak{M}(H(a), H(b), H(c)). \end{cases}$$

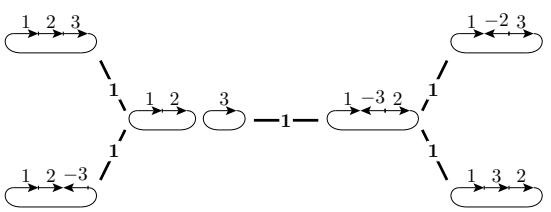
Proposing multiple candidates and then choosing the best combination is a crucial feature of our algorithm. Consider a simple example on a quartet phylogeny in Figure 7.2. The Steinerization approach may get stuck in a local optimum as in Figure 7.2(a) (both ancestral genomes are medians of the neighbouring vertices). To avoid such local optima, the Steinerization method is repeated from different starting configurations. On the other hand, if we consider the neighbouring genomes (that are within one DCJ operation from the current ancestors) and then choose the best combination, we obtain a better solution, shown in Figure 7.2(b).

We can generalize this example to configurations that will result in arbitrarily bad local optima of the Steinerization method, whereas the global optimum can be found by our method.

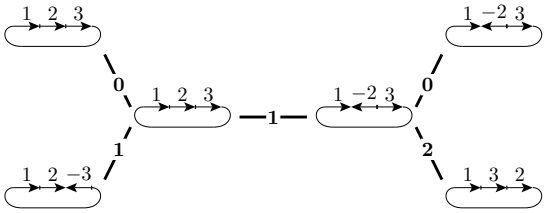
7.2.1 Finding the Best History in a Neighbourhood

Even though the size of the neighbourhood $\mathfrak{N}(H)$ can be exponential (it has $\prod_v |\,\text{cand}(H, v)|$ elements), the best history can be found in polynomial time using dynamic programming.

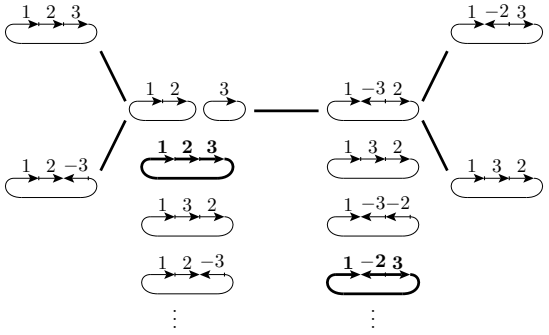
Let c_i^u be the i-th candidate from $\text{cand}(H, u)$ and let $M[u, i]$ be the lowest score, we can achieve for the subtree rooted at u if we choose candidate c_i^u as an ancestor. $M[u, i] = 0$ if u is a leaf. If u is an



(a) The Steinerization method can get stuck in this local optimum.



(b) The global optimum for the given extant genomes.



(c) Our method proposes various candidates for each ancestral genome and then takes the best combination.

Figure 7.2: A simple example showing a situation, where our more general approach outperforms the Steinerization method. Given a quartet phylogeny and genomes at the leaves, the Steinerization method has a local optimum with score 5 under the DCJ model (a), while there is a better solution with score 4 (b) which may be obtained when considering multiple candidates and choosing their best combination (c).

internal vertex with children v and w, we first compute values $M[v,j]$, $M[w,k]$ for all j,k. Then

$$M[u,i] = \min_j\{M[v,j] + d(c_i^u, c_j^v)\} + \min_k\{M[w,k] + d(c_i^u, c_k^w)\}.$$

This algorithm can be easily generalized for non-binary phylogenetic trees.

If n is the number of species, m is the number of markers in each genome, and k is the number of candidates for each ancestor, the best history can be found in time $O(nmk^2)$ (provided that the distance between two genomes can be computed in $O(m)$ time).

7.2.2 Strategies for Proposing Candidates

A crucial part of our method is proposing good candidates. By proposing more candidates, we explore a larger neighbourhood, but finding the best combination of candidates is slower. Furthermore, if we propose only candidates that are close to the genomes in the current history, the convergence to the local optimum may be slow. Here, we list several strategies for proposing candidates.

Extant species. In the initialization step, we can take genomes of the extant species as candidates in each internal node to get an evolutionary history to start with.

Intermediates. For a vertex v with adjacent vertices u and w, we can take intermediate genomes as candidates, i.e. if π, γ are genomes at u and w, we can sample genomes θ such that $d(\pi, \theta) + d(\theta, \gamma) = d(\pi, \gamma)$.

Medians. The Steinerization method uses a median of the genomes in the three adjacent vertices as a candidate. Note that often there are many medians with the same score. Furthermore, Eriksen (2007) shows that medians of moderately distant genomes may be spread wide apart. In our method, we do not need to decide beforehand which median to use. Instead, we consider all the medians as candidates, as already advocated by Eriksen (2009) and Bernt et al. (2007) in `amGRP` (`amGRP`, however, backtracks over different choices).

If we compute the median by branch-and-bound, the time to list all medians is comparable to the time to find just one median (median solvers of Siepel and Moret (2001), and Caprara (2001) are capable of listing all medians). If we try to find the median heuristically, by repeatedly moving the given genomes closer to each other, we can take the intermediate genomes as candidates. Another option is to find just a single median and search its neighbourhood for medians which can be added to the candidate list.

Neighbours. We can include neighbourhoods of individual genomes. In particular, if $H(v) = \pi$, we can add the set $\mathfrak{N}(\pi) = \{\gamma \in \mathcal{G} \mid d(\pi, \gamma) \leq 1\}$ to $\text{cand}(H, v)$. For most models, the size of $\mathfrak{N}(\pi)$ is roughly quadratic in the number of markers. Since for large genomes this becomes infeasible, we may include only genomes that do not increase the total distance to adjacent vertices, genomes closer to some genome in an adjacent vertex, or focus on a particular subset of neighbours.

Best histories. We can take several locally optimal histories and use the reconstructed ancestors as candidates. In this way we can "recombine" locally optimal solutions discovered previously.

Linearization. In Chapters 2 and 6, we tried to describe various genome models as restricted versions of the more general DCJ model. When working with linear (or multilinear) genomes, we first relax the linearity constraint and optimize the score over all mixed genomes. Then, starting from these local optima, we propose similar linear genomes and optimize in the more restricted model.

7.2.3 Unequal Gene Content

A useful extension of our method is to allow a set of possible genomes in each leaf to be given on input instead of one fixed genome $\pi(v)$. This feature is useful if we are uncertain about the order of markers in some genomes. The algorithm will choose one of the alternative gene orders, so as to minimize the overall parsimony cost. Note that this choice can change between the iterations, and consequently we do not commit to a particular interpretation of the dataset until the end of the local optimization.

In addition to modeling uncertainty about the gene order in the extant species, we can also use this method for handling recent duplications or losses. Genome rearrangement models usually require equal gene content in all considered genomes. However, if one of the genomes contains a duplicated segment of markers, we can try to delete one or the other copy, producing two alternative gene orders that are used as candidates for the corresponding leaf of the tree. The algorithm will choose one of them for the

Table 7.1: The number of operations used to explain *Campanulaceae* dataset under different models and with different algorithms.

	reversal distance	unichr. DCJ	general DCJ
`GRAPPA` (Moret et al., 2001a)	67		
`MGR` (Bourque and Pevzner, 2002)	65		
`GRAPPA` (Moret et al., 2002b)	64		
`BADGER` (Larget et al., 2005)	64		
`ABC` (Adam and Sankoff, 2008)		64	**59**
`GASTS` (Xu and Moret, 2011)		63	
`PIVO` (Kováč et al., 2011a)	**62**	**62**	**59**

locally optimal history H, presumably the one corresponding to the ancestral state before the duplication happened. We can extend this idea and use a larger set of candidates in case of multiple duplications or a gene loss. However, list of candidates will become prohibitively large for genomes with many such events.

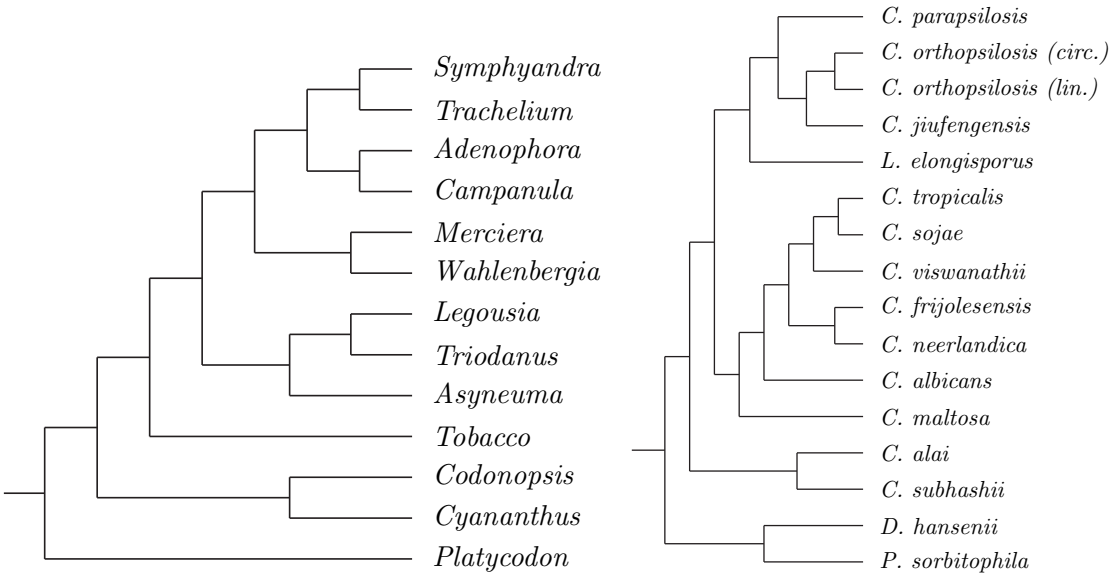
7.3 Results

We have implemented our new method and the strategies for exploring neighbourhoods described in the previous section using the DCJ and reversal rearrangement models. We demonstrate utility of our method on two datasets.

The *Campanulaceae* cpDNA dataset. For comparison, we applied our program to a well-studied dataset of 13 *Campanulaceae* chloroplast genomes (Cosner et al., 2000). Each genome in this dataset consists of a circular chromosome with 105 markers. Using the phylogenetic tree in Fig. 7.3(a) reconstructed by Bourque and Pevzner (2002) with `MGR`, the results are presented in Table 7.1.

Using `GRAPPA` software, Moret et al. (2001a) found 216 tree topologies and evolutionary histories with 67 reversals. Bourque and Pevzner (2002) using `MGR` later found a solution with 65 reversals. Even better solutions with 64 reversals were found by Moret et al. (2002b) (using `GRAPPA`) and Larget et al. (2005) (using `BADGER`).

Adam and Sankoff (2008) used the more general DCJ model and the phylogenetic tree by Bourque and Pevzner. They found a history with 64 DCJ operations with ancestors having a single chromosome, and a history with 59 DCJ operations with unconstrained ancestors. However, as Adam and Sankoff note: "There is no biological evidence in the *Campanulaceae*, or other higher plants, of chloroplast genomes consisting of two or more circles." The additional circular chromosomes are an artifact of the DCJ



(a) Phylogenetic tree of *Campanulaceae*

(b) Phylogenetic tree of *Candida*

Figure 7.3: Phylogenetic trees used in the experiments.

model, where a transposition or a block interchange operation can be simulated by circular excision and reincorporation.

Even better result for the DCJ model was obtained by Xu and Moret (2011) with their new software `GASTS`. Parenthetically, we remark that the result of Xu and Moret (2011) and our own result were coincidentally presented at the same conference.

In our program, we first tried to solve the unconstrained problem. Then we penalized multiple chromosomes in the dynamic programming objective function to avoid additional circular chromosomes and found several histories with 62 DCJ operations, where all the ancestors were unichromosomal. Moreover, these histories only require 62 reversals, which further improves on the best previously known result of 64 reversals by Moret et al. (2002b) and Larget et al. (2005).

The *Hemiascomycetes* mtDNA dataset. We have also studied evolution of gene order in 16 mitochondrial genomes of pathogenic yeasts from the 'CTG' clade of *Hemiascomycetes* (Valach et al., 2011). The phylogenetic tree (Fig. 7.3(b)) was calculated by `MrBayes` (Ronquist and Huelsenbeck, 2003) from protein sequences of 14 genes and is supported by high posterior probabilities on most branches.

The genomes consist of 25 markers: 14 protein-coding genes, two rRNA genes, and 24 tRNAs. Several challenges make this dataset difficult. First, it combines genomes with a variety of genome architectures: *C. subhashii, C. parapsilosis*, and *C. orthopsilosis* are linear, *C. frijolesensis* has two linear chromosomes, and the rest of the species have circular-mapping chromosomes.

Some of the genomes (*C. albicans, C. maltosa, C. sojae, C. viswanathii*) contain recent duplications which cannot be handled by the DCJ model. As outlined in Section 7.2.3, we have removed duplicated

genes, and included both possible forms of the genomes as alternatives in the corresponding leaves. Similarly, the genomes of *C. alai, C. albicans, C. maltosa, C. neerlandica, C. sojae*, and *L. elongisporus* contain long inverted repeats that are often subject to recombination resulting in reversal of the portion of the genome between the two repeats. Both forms of the genome are routinely observed in the same species, and we include both of them in the corresponding leaf.

Finally, we penalized occurrences of multiple circular chromosomes and combinations of linear and circular chromosomes in ancestral genomes. Such combinations would likely represent artifacts of the DCJ model.

Our algorithm, using the *extant species, neighbours,* and *best histories* strategies, has found an evolutionary history with 78 DCJ operations. More detailed discussion of this dataset (including comparison to manual reconstruction in a subtree of closely related species) is included elsewhere (Valach et al., 2011).

7.4 Conclusion

We have developed a new method for reconstructing evolutionary history and ancestral gene orders, given the gene orders of the extant species and their phylogenetic tree. We have implemented our method using the double cut and join model and studied evolution of gene order in 16 mitochondrial yeast genomes, demonstrating applicability of our approach to real biological datasets. We have also analyzed the thoroughly studied *Campanulaceae* dataset and improved upon the previous results (Moret et al., 2001a; Bourque and Pevzner, 2002; Moret et al., 2002b; Larget et al., 2005; Adam and Sankoff, 2008).

Our framework is compatible with a variety of rearrangement models and the optimization can be adjusted by introducing new strategies of generating candidate ancestral genomes. The use of the DCJ allowed us to study datasets that contained both linear and circular genomes and to contribute towards understanding mechanisms of genome linearization during the evolution (Valach et al., 2011).

In our experiments, we have explored only a small fraction of possible strategies offered by our framework. Systematic study of new initialization methods, candidate sets, and rearrangement measures may lead to even better results for a variety of practical problems. Our work also opened an avenue towards a systematic solution of the problems with unequal gene content. Similar directions can perhaps lead to the possibility of incorporating incompletely assembled genomes, a challenge posed by the next-generation sequencing technologies.

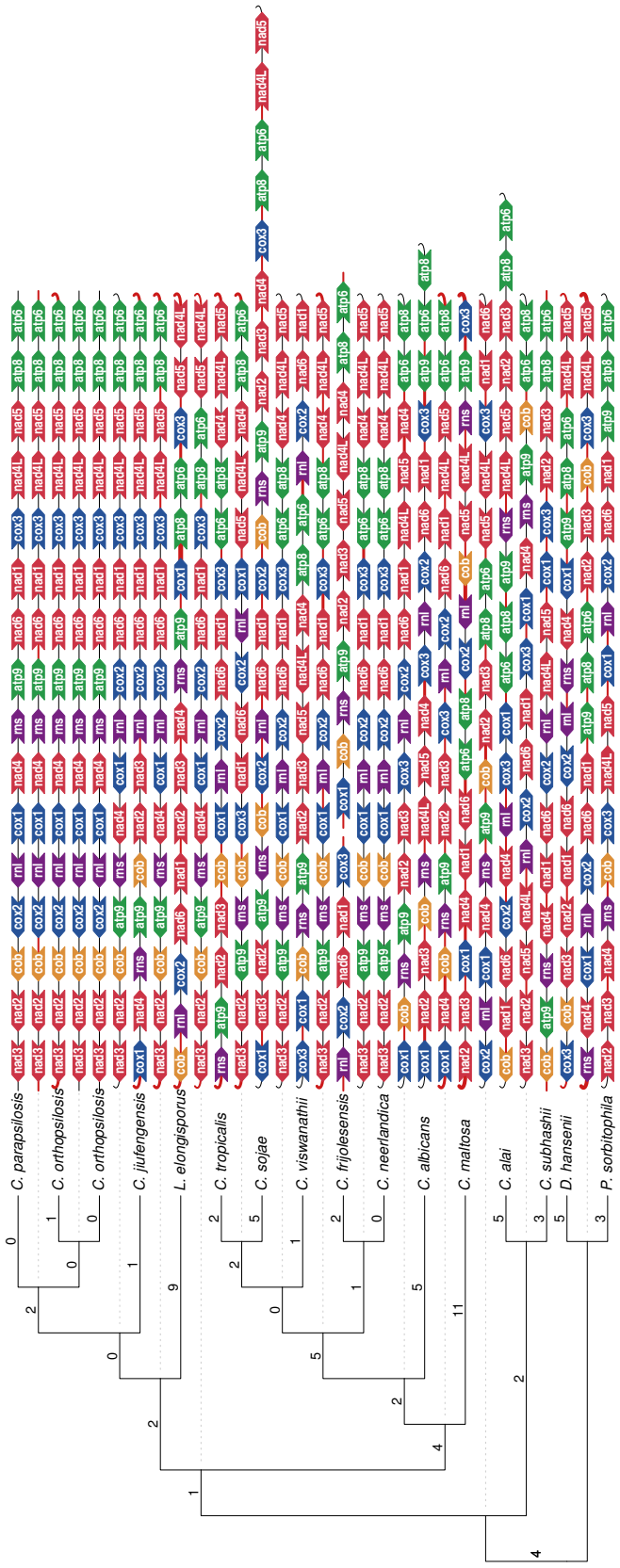


Figure 7.4: Reconstructed evolutionary history for the mitochondrial genomes of the *Hemiascomycetes* yeasts from Fig. 7.1. Numbers on the edges represent the number of DCJ operations from each genome to its ancestor.

Chapter 8

Complexity of rearrangement problems under the breakpoint distance

8.1 Introduction

In this chapter, we study rearrangement problems in different variants of the breakpoint model and settle several open questions regarding their computational complexity. In Chapter 3, we studied the problem of reconstructing a single ancestral gene order, the so called median problem. The ultimate goal is reconstructing evolutionary history of large phylogenies, alas, we have seen that in most of the genome models (DCJ, restricted DCJ, reversal, unichromosomal or multilinear breakpoint models), even the median problem is NP-hard.

One notable exception was the general breakpoint model (see Section 3.1.2). Tannier et al. (2009) observed that if we drop the condition that genomes are unichromosomal and that all chromosomes are linear, we get a very simple model where the median problem is solvable in polynomial time. Even though this model is not biologically plausible and more realistic models exist, the breakpoint model may still be useful for upper and lower bounds, and solutions in this model may serve as good starting points for more elaborate and complicated models.

Two interesting open questions remained in the work of Tannier et al. (2009). These are also articulated in the monograph by Fertin et al. (2009):

1. The best time complexity for the median and guided halving problems under the breakpoint distance on multichromosomal genomes (with circular chromosomes allowed) is $O(n^3)$, using a reduction to the maximum weight perfect matching problem. It is an open problem to devise an ad-hoc algorithm with better complexity.

2. The small parsimony and large parsimony problems under the breakpoint distance are open regarding multichromosomal signed genomes where linear and circular chromosomes

Table 8.1: Our new results in context of the previously known results. *Tannier et al. (2009) †Zheng et al. (2008) ‡Pe'er and Shamir (1998); Bryant (1998)

Breakpoint Model	Median	Halving	Guided Halving	Small Phylogeny
unichromosomal (linear or circular)	NP-hard ‡	NP-hard [new]	NP-hard †	NP-hard [trivial]
multilinear	NP-hard *	NP-hard [new]	NP-hard †	NP-hard [trivial]
multichromosomal (circular or mixed)	$O(n^3)$ $O(n\sqrt{n})$ [new]	$O(n^3)$ * $O(n)$ [new]	$O(n^3)$ * $O(n\sqrt{n})$ [new]	NP-hard [new]

are allowed.

We resolve the first question in a positive way by showing a more efficient algorithm running in $O(n\sqrt{n})$ time. This is by reduction to the maximum cardinality matching problem. Moreover, we show that maximum cardinality matching can be reduced back to the breakpoint median (by a linear reduction) and so the two problems have essentially the same complexity. The same technique also improves the algorithms for halving and guided halving.

The second question is resolved in a negative way. One could expect that the large parsimony problem is NP-hard for this model, since it is NP-hard even for the Hamming distance on binary strings (Foulds and Graham, 1982). However, surprisingly, for the breakpoint distance (unlike for the Hamming distance), the small phylogeny is NP-hard (and APX-hard) even for four species, i.e., a quartet phylogeny. In other words, while finding an ancestor for three species is easy, finding two ancestors for four species is already hard.

Apart from the general breakpoint model, we also study the unichromosomal and the multilinear breakpoint model. Tannier et al. (2009) conjectured that the halving problem is tractable in these models – after all, the halving problem can be solved in linear time in more complex models such as DCJ or RT. We refute this conjecture by showing that, in the unichromosomal and the multilinear breakpoint model, the halving problem is NP-hard. Curiously, this is the first known rearrangement problem that is harder in the breakpoint model than in the DCJ or reversal models.

The previous work and our new results are summarized in Table 8.1.

Note on breakpoint distance and similarity. In this chapter, we will work with the general breakpoint model as introduced in Section 2.1. Recall that the breakpoint distance is defined as

$$bp(\pi, \gamma) = n - \mathrm{sim}(\pi, \gamma),$$

where n is the number of genes and

$$\mathrm{sim}(\pi, \gamma) = a(\pi, \gamma) + \frac{e(\pi, \gamma)}{2}$$

is the number of common adjacencies plus half the number of common telomeric adjacencies.

The breakpoint distance satisfies all properties of a metric and is used in the literature, however, we find it easier to work directly with the similarity measure $\mathrm{sim}(\pi_1, \pi_2)$. Instead of minimizing the sum of distances, we will try to maximize the sum of similarities, i.e., maximize the number of common adjacencies.

Similarly, recall that the double distance in the breakpoint model can be computed according to the formula

$$dd_{bp}(\pi, [\delta]) = 2n - \mathrm{sim}(\pi, [\delta]),$$

where we define $\mathrm{sim}(\pi, [\delta])$ as

$$\mathrm{sim}(\pi, [\delta]) = a(\pi, [\delta]) + \frac{e(\pi, [\delta])}{2}.$$

Here, $a(\pi, [\delta])$ is the number of adjacencies in common and $e(\pi, [\delta])$ the number of telomeric adjacencies in common, while adjacencies twice in common are counted as 2.

Road map. In the next section, we refute the conjecture of Tannier et al. (2009) and prove that the halving problem is NP-hard for the unichromosomal and multilinear breakpoint model. In the following two sections, we study the general breakpoint model. In Section 8.3, we look at the median problem: we improve upon the algorithm of Tannier et al. (2009) and show that it is equivalent to the maximum matching problem. The hardness of the small phylogeny problem is studied in Section 8.4 and we give concluding remarks in Section 8.5.

These results were presented at the RECOMB-CG 2012 conference and a paper has been accepted for publication in the *Journal of Computational Biology*.

8.2 Halving Problem

Bryant (1998) showed that the median problem is NP-hard in the circular breakpoint model by reduction from the Directed-Hamiltonian-Cycle problem. The halving problem was not studied previously in the breakpoint model, but we show that it suffers the same "Hamiltonian" curse as the median problem – in order to find the ancestor, we would in fact have to find a Hamiltonian cycle. Our proof is even simpler than that of Bryant (1998).

As the halving problem is polynomially solvable in more realistic models such as the RT model (El-Mabrouk and Sankoff, 2003) or the DCJ model (Alekseyev and Pevzner, 2007b; Mixtacki, 2008; Warren and Sankoff, 2009b; Kováč et al., 2011b), the halving problem under the breakpoint distance will remain a mere curiosity: It is the first problem which is easier in the DCJ or even in the RT model than in the breakpoint model. Furthermore, it is the only known case where halving is NP-hard, while the double distance is computable in polynomial time (e.g., in the DCJ model, the opposite is true – halving is easy, while the double distance is NP-hard (Tannier et al., 2009)).

Curiously, the ordinary halving problem was not studied before in the breakpoint model, and Tannier et al. (2009) also leave it open. Moreover, they conjecture that the problem is polynomially solvable – this might perhaps be attributed to the fact that the halving problem is polynomially solvable in far more complicated models such as reversal/translocation (RT) (El-Mabrouk and Sankoff, 2003) or double cut and join (DCJ) (Alekseyev and Pevzner, 2007b; Mixtacki, 2008; Warren and Sankoff, 2009b; Kováč et al., 2011b). Nevertheless, we refute this conjecture (unless P = NP) by proving that the halving problem is NP-complete in the unichromosomal and multilinear models.

Theorem 13. *Halving problem is NP-hard in the circular, linear, and multilinear breakpoint models.*

Proof. The proof is by reduction from the Directed-Hamiltonian-Cycle problem. Plesník (1979) proved that this problem is still NP-hard for graphs with maximum degree 2 and the construction implies the problem is also NP-hard if all in-degrees and out-degrees are equal to 2. Note that such graphs have an Eulerian cycle.

Let $G = (V, E)$ be such directed graph; the corresponding doubled genome δ will have two copies of a gene for each vertex in G and an Eulerian cycle in G traversing each vertex twice will be the order of genes in δ. More precisely, let $G' = (V', E')$, where $V' = \{x_1^+, x_1^-, x_2^+, x_2^- : x \in V\}$ and the edges in E' are defined as follows: traverse the Eulerian walk and for each edge $xy \in E$, include edge $x_i^+ y_j^-$ in E', where i and j is 1 if we are visiting the vertex for the first time, and 2 if we are visiting the vertex for the second time. Note that all edges go from head to tail, E' is a perfect matching, and G' defines the doubled genome δ consisting of a single circular chromosome.

Let α be a circular genome that is a solution to the halving problem. Note that δ has no double adjacencies, so α can have at most n adjacencies in common (none twice in common). This maximum can be attained if and only if all the adjacencies in α are of the form $x^+ y^-$ (from head to tail) and for each such adjacency, $x_i^+ y_j^-$ is an adjacency in δ for some i, j. This is true if and only if $xy \in E$. So by contracting the base matching (each head and tail of a gene into a single vertex) and orienting the edges (from head to tail), we get a directed Hamiltonian cycle in G.

For the linear and multilinear models, remove one edge xy from G and consider the problem of deciding whether G contains a directed Hamiltonian path. This problem is still NP-hard and can be reduced to the halving problem in the linear models: G now has an Eulerian path starting in y and ending in x. We replace the last adjacency $x_2^+ y_1^-$ in δ (corresponding to the removed edge) by two telomeric adjacencies $x_2^+ \mathrm{T}_{x_2^+}$ and $y_1^- \mathrm{T}_{y_1^-}$ to get a linear genome. If α is a linear or multilinear solution to the halving problem, it can reach the maximum similarity if and only if all of its adjacencies (including the telomeric adjacencies) are in common with δ and this is true if and only if contraction of α is a directed Hamiltonian path in G. □

8.3 Median and Halving Problems in the General Model

From now on, we will study the *general* breakpoint model, i.e., the multichromosomal circular model where genomes are perfect matchings. We will also note how to extend the results to the mixed model and use the developed techniques for halving and guided halving problems.

8.3.1 Breakpoint Median

Tannier et al. (2009) noticed that finding a breakpoint median can be reduced to finding a maximum weight perfect matching. This can be done in $O(n^3)$ time by algorithm of Gabow (1973) and Lawler (1976). An open problem from Tannier et al. (2009) and Fertin et al. (2009) asks, whether this can be improved. We answer this question affirmatively by showing an $O(n\sqrt{n})$ algorithm.

The solution by Tannier et al. (2009) (if we rephrase it using the similarity measure instead of the breakpoint distance) was to create a complete weighted graph G where vertices are extremities and weight $w(xy)$ of edge xy is the number of genomes which contain the adjacency xy. Any perfect matching α corresponds to some genome and the weight of the matching is equal to its median score $S(\alpha)$.

Notice that instead of finding a maximum weight *perfect* matching, we can remove all the zero-weight edges from G and find an ordinary (not necessarily perfect) matching. We can then complete the genome by joining the free vertices arbitrarily. Since the number of edges in G is now linear, maximum weight matching can be found in $O(n^2 \log n)$ time by algorithm of Gabow (1990) or even in $\tilde{O}(n\sqrt{n})$ time by the state of the art algorithm of Gabow and Tarjan (1991) using the fact that the weights are small integers. More generally and more precisely:

Theorem 14. *The* Breakpoint-Median *problem can be solved in* $O(kn\sqrt{n} \cdot \log(kn)\sqrt{\alpha(kn, n) \log n})$ *time for* k *genomes in the general model. (Here,* $\alpha(m, n)$ *is the inverse Ackermann function.)*

We further improve the algorithm for the most important special case, $k = 3$: Notice that when xy is an edge with weight 3, there is no other edge incident to x or y. Therefore, xy must belong to the maximum weight matching. Moreover, if xy has weight 2, there is a maximum weight matching which contains xy. Suppose to the contrary that xu and yv were matched in α instead. Then $w(xu)$ and $w(yv)$ is at most 1 and by exchanging these edges for xy and uv with weights $w(xy) = 2$ and $w(uv) \geq 0$, we get a matching with the same or even higher weight.

Thus, we can include all edges of weight 2 and 3 in the matching and remove matched vertices together with their incident edges. The remaining graph has only unit edge weights, so it suffices to find maximum *cardinality* matching. This can be done in $O(m\sqrt{n})$ time by the algorithm of Micali and Vazirani (1980). Thus, we have the following claim.

Theorem 15. *The* Breakpoint-Median *problem for three genomes can be solved in* $O(n\sqrt{n})$ *time (in the general model).*

One might still wonder whether there is an even better algorithm for the median problem, which perhaps can avoid computation of maximum matching. Alas, we show that improving upon our result would be very hard, since it would immediately imply a better algorithm for the matching problem, beating the result of Micali and Vazirani (1980) (at least on cubic graphs), which has been an open problem for more than 30 years.

Biedl (2001) showed that the maximum matching problem is reducible to the maximum matching problem in *cubic* graphs by a *linear* reduction. This means that we can transform any given graph G with m edges to a cubic graph G' with $O(m)$ edges such that the maximum matching in G can be recovered from one in G' in $O(m)$ time. Thus, any $O(f(m))$ algorithm for finding maximum matching in cubic graphs implies an $O(f(m)+m)$ algorithm for arbitrary graphs.

We say that a reduction is *strongly linear* if it is linear, and both the number of vertices and the number of edges increase at most linearly. Such a reduction preserves the running time $O(f(m,n))$ depending on both the number of vertices and the number of edges.

We prove that the Breakpoint-Median problem is equivalent to Matching under linear reduction and to Cubic-Matching under strongly linear reduction. If we write $\leq_\ell$ for linear and $\leq_{s\ell}$ for strongly linear reduction, we have

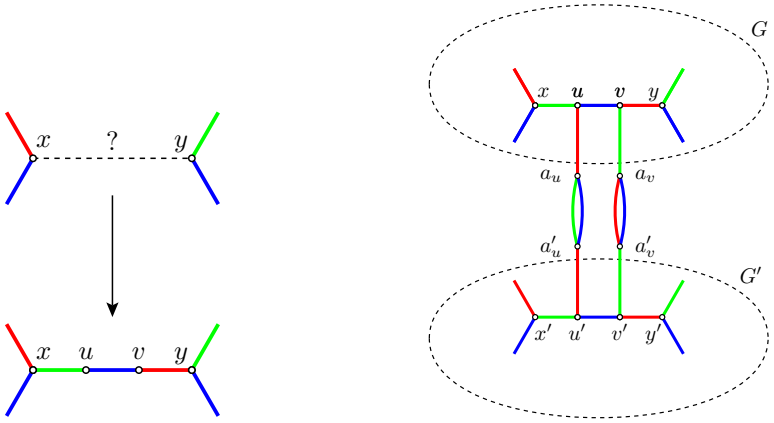
$$\textsc{Matching} \leq_\ell \textsc{Cubic-Matching} \leq_{s\ell} \textsc{Breakpoint-Median} \leq_{s\ell} \textsc{Matching}.$$

The first reduction is by Biedl (2001) and the last one was shown in Theorem 15 (in fact, a reduction to Subcubic-Matching, where the degrees are at most 3, was shown – this is equivalent to Cubic-Matching under the strongly linear reduction (Biedl, 2001)). We now prove the middle reduction.

Let G be a cubic graph, an instance of the Cubic-Matching problem. The difference between the Cubic-Matching and Breakpoint-Median problem is that in Breakpoint-Median, the input multigraph consists of three perfect matchings, i.e., is edge 3-colourable. However, not all cubic graphs are edge 3-colourable (take for example Petersen's graph).

The solution is to colour edges arbitrarily and resolve conflicts as shown in Figure 8.1(a). We can, for example, colour the ends of edges at each vertex randomly by three different colours. When both ends of an edge are assigned the same colour, we colour the edge appropriately. When the ends have different colours, we subdivide the edge into three parts and use the third colour for the middle edge (see Figure 8.1(a)). Note that the size of a maximum matching in the modified graph is exactly one more than the size in the original graph: If xy is matched in the original, xu and vy can be matched in the modified graph. If xy is not matched, we can still match uv.

Now, the modified graph is edge 3-colourable but not cubic. We remedy this by duplicating the whole graph and connecting the corresponding vertices of low degree as shown in Figure 8.1(b). As noted above, we may suppose that the auxiliary double edges $a_u a'_u$ and $a_v a'_v$ are matched, so ua_u, $u'a'_u$, va_v, and $v'a'_v$ are not matched and given the solution for the Breakpoint-Median problem, we can



(a) Edge xy (top) should be coloured green (this is the only missing colour at x) and red at the same time (this is the missing colour at y). We resolve this conflict by subdividing edge xy by two new vertices (bottom); we colour xu green, vy red and uv blue.

(b) In the second phase, we duplicate graph G and connect the corresponding vertices with degree 2 as shown in the figure.

Figure 8.1: Linear reduction of maximum matching in cubic graphs to breakpoint median problem.

recover the maximum matching of G in $O(n)$ time. The reduction is obviously linear, so we have the following claim.

Theorem 16. *The* Breakpoint-Median *problem (in the general model) has the same complexity as finding maximum cardinality matching in cubic graphs.*

8.3.2 Median in the Mixed Model

In the mixed model, weight of a telomeric adjacency $x\mathrm{T}_x$ is equal to half the number of genomes that contain $x\mathrm{T}_x$. If we multiply all weights by 2, we can use the algorithm by Gabow and Tarjan (1991) for integer weights, so the result of Theorem 14 remains valid also in the mixed model.

For the median of three genomes, an $O(n\sqrt{n})$ algorithm exists: We observed that we can include all the double and triple adjacencies in the matching. This is also true for the double and triple telomeric adjacencies (edges of weight 1 and 1½): If $w(x\mathrm{T}_x) = 1½$, $x\mathrm{T}_x$ is a triple adjacency and no other edge is incident to neither x nor T_x in G. If $w(x\mathrm{T}_x) = 1$ but the median α contains adjacency xy instead, then $w(xy) \leq 1$ and since T_x can only be incident to x, it must be unmatched (or matched by a zero-weight edge) and so we can replace xy by $x\mathrm{T}_x$ in α.

The remaining graph consists of edges with unit weight and weight ½. Note however that all the ½-weight edges are of the form $x\mathrm{T}_x$, and there is no other edge incident to T_x. We use the doubling trick again: we take two copies of graph G, and replace all pairs $x\mathrm{T}_x$, $x'\mathrm{T}'_x$ by a single edge xx' of unit weight. We can then remove all the telomere vertices. The resulting graph will have only unit weight edges and the maximum matching will be exactly twice the size of the maximum matching in the original

graph.

8.3.3 Halving Problems in the General Model

The same tricks can be used for halving and guided halving problems. Recall that in the halving problem, we are given a duplicated genome γ, and we are searching for genome α that minimizes the double distance $dd(\alpha, \gamma)$; in the guided halving problem, we are also given genome ρ and we are minimizing the sum $dd(\alpha, \gamma) + d(\alpha, \rho)$.

Again, we construct graph G, where this time, weight of edge xy is the number of adjacencies among x_1y_1, x_1y_2, x_2y_1, x_2y_2 in γ and possibly xy in ρ (in case of the guided halving problem). The rest of the solution is identical, leading to an $O(n\sqrt{n})$ algorithm for the guided halving problem. In the halving problem, the degrees of vertices in G are at most 2 and after including all the double edges in the solution, the remaining graph consists only of cycles and the maximum matching can be found trivially in linear time.

8.4 Breakpoint Phylogeny

In the Small-Phylogeny problem, we try to reconstruct ancestral genomes given a phylogenetic tree and gene orders of the extant species while minimizing the sum of distances along the edges of the tree. This problem is NP-hard for most rearrangement distances and for most models; this follows trivially from the NP-hardness of the Median problem. However, as we have seen in the previous section, this is not the case in the general breakpoint model and the complexity of the Small-Phylogeny problem remained open (Tannier et al., 2009; Fertin et al., 2009).

In this section, we prove that the Small-Phylogeny problem is NP-hard also in the general breakpoint model. We show that the problem is NP-hard already for four species, a special case that we call the Breakpoint-Quartet problem.

Given four genomes $\pi_1, \pi_2, \pi_3, \pi_4$, the Breakpoint-Quartet problem is to find ancestral genomes α_1, α_2 that maximize the sum of similarities along the edges of the quartet tree (see Fig. 8.2. In other words, the sum

$$S(\alpha_1, \alpha_2) = \mathrm{sim}(\pi_1, \alpha_1) + \mathrm{sim}(\pi_2, \alpha_1) + \mathrm{sim}(\alpha_1, \alpha_2) + \mathrm{sim}(\alpha_2, \pi_3) + \mathrm{sim}(\alpha_2, \pi_4)$$

should be maximized.

Theorem 17. *The* Breakpoint-Quartet *problem is NP-hard and even APX-hard in the general breakpoint model.*

The proof is inspired by the work of Dees (2009) who showed that the following problem is NP-hard: Given two graphs $G_1 = (V, E_1)$, $G_2 = (V, E_2)$, find two perfect matchings $M_1 \subseteq E_1$ and $M_2 \subseteq E_2$ with the maximum overlap $M_1 \cap M_2$. The problem is NP-hard even when the components in G_1 and G_2 are

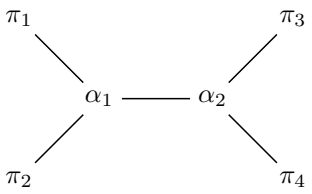


Figure 8.2: Quartet tree.

just cycles. In our proof, $\pi_1 \cup \pi_2$ will correspond to E_1, $\pi_3 \cup \pi_4$ will correspond to E_2, and the unknown ancestors α_1, α_2 will correspond to the unknown perfect matchings M_1, M_2.

Our proof is, however, much more involved and there are two reasons for this. First, the problem formulation does not guarantee that $\alpha_1 \subseteq \pi_1 \cup \pi_2$ and $\alpha_2 \subseteq \pi_3 \cup \pi_4$. We will say that solution α_1, α_2 that satisfies this condition is in a *normal form*. The hard part of the proof is showing that we can transform any solution α_1, α_2 into a solution α_1', α_2' that has the same score and moreover it is in the normal form.

The second major difficulty is that we are maximizing the sum $S(\alpha_1, \alpha_2)$ instead of just the size of the intersection. A solution with maximum score $S(\alpha_1, \alpha_2)$ does not necessarily maximize the term $\mathrm{sim}(\alpha_1, \alpha_2)$, the size of the intersection. To overcome these difficulties, we had to modify the edge gadget from the original proof and use a more restricted problem for the reduction.

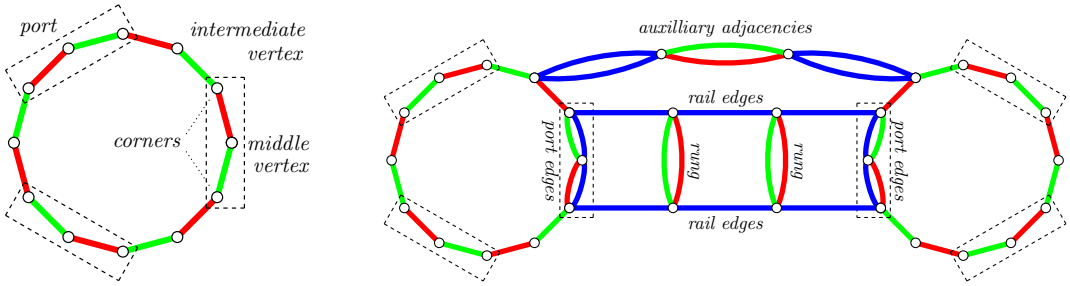
8.4.1 Overview of the Proof

The proof is by reduction from the Cubic-Max-Cut problem. Given a graph G, the Max-Cut problem is to find a cut of maximum size. We may rephrase this as a problem of colouring all vertices in G with two colours, red or green, while maximizing the number of red-green edges. (Partition of V into the red part and the green part defines a cut and its size is the number of edges with endpoints of different colour.) In the Cubic-Max-Cut problem, the instances are cubic graphs; this variant is still NP-hard and APX-hard (Alimonti and Kann, 1997).

Let $G = (V, E)$ be a given cubic graph, an instance of the Cubic-Max-Cut problem. We will construct genomes π_1, π_2, π_3, and π_4 such that the maximum cut in G can be recovered from the solution α_1, α_2 of the Breakpoint-Quartet problem in polynomial time.

For each vertex of G, there will be a vertex gadget (see Figure 8.3(a)) made of adjacencies of π_1 and π_2. Let π_1 be the red matching and π_2 the green matching. As we will prove later, we may suppose that $\alpha_1 \subseteq \pi_1 \cup \pi_2$, so within each vertex gadget, α_1 will contain either the red edges of π_1 or the green edges of π_2. This naturally corresponds to a red/green vertex colouring in the Cubic-Max-Cut problem.

The framed vertices in Figure 8.3(a) are called "ports" – this is where the edge gadgets for the three incident edges are attached. For each edge of G, an edge gadget connecting the ports of the corresponding vertex gadgets is constructed as shown in Figure 8.3(b). The blue cycles consist of two matchings – the adjacencies of π_3 and π_4. Again, as we will prove later, we may suppose that $\alpha_2 \subseteq \pi_3 \cup \pi_4$, i.e., the second ancestor consists only of the blue edges.



(a) Vertex gadget.

(b) Edge gadget.

Figure 8.3: The vertex and edge gadgets used in our reduction and the terminology used for different types of vertices and edges. The red and green edges are the adjacencies of π_1 and π_2, respectively. The cycles made of blue edges can be decomposed into two matchings – the adjacencies of π_3 and π_4.

For future reference, let us state here again the claims to be proved in the form of a lemma:

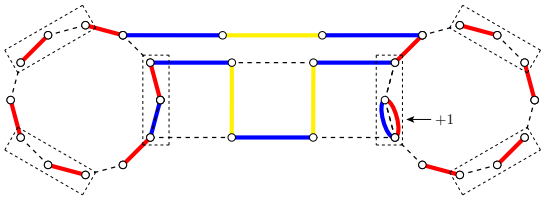
Lemma 4 (Normal form). *Let $\pi_1, \pi_2, \pi_3, \pi_4$ be an instance of the* Breakpoint-Quartet *problem constructed from a* Cubic-Max-Cut *instance as described above. Then any solution α_1, α_2 can be transformed in polynomial time into a solution α'_1, α'_2 such that $S(\alpha'_1, \alpha'_2) \geq S(\alpha_1, \alpha_2)$ and*

$$\alpha'_1 \subseteq \pi_1 \cup \pi_2 \quad \text{and} \quad \alpha'_2 \subseteq \pi_3 \cup \pi_4.$$

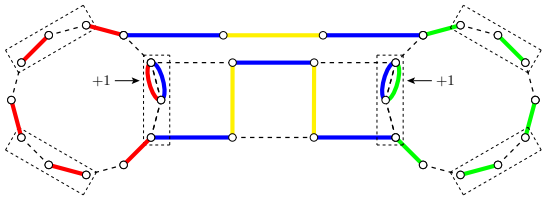
Once we prove the normal form lemma, the rest of the proof is easy: If α_1, α_2 is a solution in the normal form, term $\mathrm{sim}(\pi_1, \alpha_1) + \mathrm{sim}(\pi_2, \alpha_1)$ is always the same – we get $+6$ for each vertex gadget and $+6$ for each edge gadget. Similarly, term $\mathrm{sim}(\alpha_2, \pi_3) + \mathrm{sim}(\alpha_2, \pi_4)$ is always the same – we get $+9$ for each edge gadget. So the score $S(\alpha_1, \alpha_2)$ is maximized, when $\mathrm{sim}(\alpha_1, \alpha_2) = |\alpha_1 \cap \alpha_2|$ is maximized. Let uv be an edge in our graph G from the Cubic-Max-Cut problem. If we choose matchings of the same colour for both vertex gadgets u and v, then α_1 and α_2 can only have one edge in common within the edge gadget uv (see Figure 8.4(a)). However, if u and v have matchings of different colour, we can set adjacencies of α_2 so that α_1 and α_2 have two edges in common (see Figure 8.4(b)). When we sum up all these contributions, we get $S(\alpha_1, \alpha_2) = 20m + c$, where m is the number of edges in G, and c is the size of the cut corresponding to the matching α_1, so a polynomial algorithm for Breakpoint-Quartet would imply a polynomial algorithm for Cubic-Max-Cut.

For the APX-hardness, note that for any graph with m edges, we can easily find a cut of size $c \geq m/2$. Let α_1^*, α_2^* be an optimal solution for an instance of the Breakpoint-Quartet problem and α_1, α_2 a solution such that $S(\alpha_1^*, \alpha_2^*) \leq (1+\varepsilon) S(\alpha_1, \alpha_2)$. Let both solutions be in the normal form, and let c^* and $c \geq m/2$ be the sizes of the corresponding cuts. Then $20m + c^* \leq (1+\varepsilon)(20m + c)$, and $c^* \leq (1+\varepsilon)c + 20\varepsilon m \leq (1+41\varepsilon)c$. So a $(1+\varepsilon)$-approximation algorithm for the Breakpoint-Quartet problem would lead to a $(1+41\varepsilon)$-approximation algorithm for the Cubic-Max-Cut problem.

It can also be proved that the phylogenetic tree $((\pi_1, \pi_2), (\pi_3, \pi_4))$ is the most parsimonious. The alternative quartets $((\pi_1, \pi_3), (\pi_2, \pi_4))$ and $((\pi_1, \pi_4), (\pi_2, \pi_3))$ yield score $\leq 20m$, so this result also implies



(a) Adjacencies of the first ancestor α_1 (red edges) agree with the adjacencies of π_1 at both vertex gadgets. This corresponds to colouring both vertices red in the Cubic-Max-Cut problem. Note that α_1 and α_2 can only have one adjacency in common.



(b) In the first vertex gadget, α_1 agrees with π_1 (red edges) and in the second gadget, α_1 agrees with π_2 (green edges). This corresponds to colouring the first vertex red and the second vertex green in the Cubic-Max-Cut problem. In this case, α_1 and α_2 have two adjacencies in common.

Figure 8.4: The dashed edges indicate the underlying vertex and edge gadgets, the blue edges are adjacencies of α_2 and the red, green, and yellow edges are adjacencies of α_1. Here, we assume that α_1 and α_2 are in the normal form.

the NP- and APX-hardness of the Large-Phylogeny problem. It remains an open problem whether computing the correct quartet (without reconstructing the ancestors) is hard.

8.4.2 Notation, Terminology, and Other Conventions

We say that an adjacency $e \in \alpha_1$ is *supported* if $e \in \pi_1 \cup \pi_2$. Similarly, $e \in \alpha_2$ is *supported* if $e \in \pi_3 \cup \pi_4$. An adjacency that is not supported is *unsupported*. Furthermore, let $\Pi = \pi_1 \cup \pi_2 \cup \pi_3 \cup \pi_4$ be the set of adjacencies present in at least one extant species. We will say that an adjacency $e \in \alpha_i$ is *weakly supported*, if $e \in \Pi$.

Let us name the different types of vertices (extremities) and edges (adjacencies) in the following manner. The framed vertices in Fig. 8.3(a) are called *ports* and edges from $\pi_1 \cup \pi_2$ that connect them are called *port* edges. We use the same names also for other (extant or ancestral) adjacencies which are parallel to these.

Each port consists of two outer extremities called *corners*, and the *middle* vertex between them. The sets of all ports, corners, and middle vertices are denoted by P, C, and M, respectively ($P = C \cup M$). The set of *intermediate* extremities located between ports of vertex gadgets is denoted by I.

The double edges and the two vertices at the top of Fig. 8.3(b) are *auxiliary* – they just complete the matchings into perfect matchings.

Since the edge gadget without auxiliary and port edges reminds of a *ladder*, we use the following terminology (see Fig. 8.3(b)). The red-green double adjacencies are the *rungs* and the blue adjacencies are the *rails* of the ladder. Again, we use the same name for parallel adjacencies. The set of auxiliary extremities is denoted by A and the set of ladder extremities is denoted by L.

We say that uv is an X–Y-edge if $u \in X$ and $v \in Y$ (X and Y do not have to be disjoint); an X-edge

is any edge uv such that $u \in X$ or $v \in X$.

In the proof of the normal form lemma, we will gradually transform a given solution α_1, α_2 by exchanging some of the adjacencies in the solution for other adjacencies. The method is analogous to improving a given matching by an augmenting path. An *α_i-alternating* cycle is a cycle where edges belonging to α_i and edges not belonging to α_i alternate. We will say that C_1, C_2 is a *non-negative pair of cycles* for the solution α_1, α_2, if C_i is an α_i-alternating cycle and exchanging the matched and the unmatched edges of C_i in α_i (for $i = 1, 2$) does not decrease the score:

$$S(\alpha_1 \oplus C_1, \alpha_2 \oplus C_2) \geq S(\alpha_1, \alpha_2).$$

One of the cycles may be empty, in which case we simply say that C_1 or C_2 is a *non-negative cycle*, and if the exchange in fact increases the score, we may speak of an *augmenting* pair of cycles (or an augmenting cycle).

In the figures that follow, we will colour adjacencies of α_2 blue and adjacencies of α_1 red, green, or yellow. We use red or green for edges in the vertex gadgets that are shared with π_1 or π_2, respectively (this corresponds to choosing the red or green colour in the Cubic-Max-Cut problem). We use yellow for the other edges. We use straight lines for the actual adjacencies and wavy lines for the suggested adjacencies in non-negative cycles that should be included instead.

In the proof, we will often say

> *we may suppose that the solution has property $\mathcal{P}$*

as a shorthand for a more precise (and longer) statement

> *Given any solution α_1, α_2, we can transform it to a solution α_1', α_2' with $S(\alpha_1', \alpha_2') \geq S(\alpha_1, \alpha_2)$ having property $\mathcal{P}$ in polynomial time; in particular, if α_1, α_2 is an optimal solution, α_1', α_2' is also optimal, with property $\mathcal{P}$. From now on, we will assume that the solution has property $\mathcal{P}$.*

With this terminology, we may rephrase the normal formal lemma more succinctly as follows: *We may suppose that solutions of the instances obtained by reduction from* Cubic-Max-Cut *as described above have all adjacencies supported.*

8.4.3 Proof of the Normal Form Lemma

First, we focus on the adjacencies that the ancestors α_1 and α_2 have in common. We will show that these may be assumed to be at least weakly supported.

Proposition 1. *We may suppose that all red-green double edges (auxiliary adjacencies and rungs) are matched in α_1 and all blue double edges (auxiliary adjacencies) are matched in α_2, i.e., $\pi_1 \cap \pi_2 \subseteq \alpha_1$ and $\pi_3 \cap \pi_4 \subseteq \alpha_2$.*

Proof. We can alternately replace genome α_1 or α_2 by the median of its neighbours in the phylogenetic tree until we converge to a local optimum. As we have already proved in the previous section, we may assume that a median contains all adjacencies occurring at least twice. □

Proposition 2. *We may suppose that α_1 and α_2 do not contain unsupported M-edges. In other words, we may suppose that in both α_1 and α_2, one of the edges in each port is chosen.*

Proof. Let $x \in M$. First, consider the case that $xy_1 \in \alpha_1$ and $xy_2 \in \alpha_2$ are both unsupported. Let p be a neighbouring corner vertex. While xy_1 and xy_2 contribute at most $+1$ to the score (if $y_1 = y_2$), common adjacency xp would contribute $+3$. Let pz_1 and pz_2 be the actual adjacencies in α_1 and α_2; either $z_1 \neq z_2$, or $z_1 = z_2$ and one of the adjacencies is unsupported. Either way, these two edges contribute at most $+2$ to the score; so xpz_1y_1x and xpz_2y_2x is a non-negative pair of cycles and we can exchange the edges.

Similarly, if one ancestor contains a port edge xp and the other one adjacencies pz and unsupported xy, then $xpzyx$ is a non-negative cycle. □

Proposition 3. *We may suppose that all L-edges are weakly supported – they are ladder edges.*

Proof. In α_1, all L-edges are the rung edges by Proposition 1 and are supported. Consequently, contribution of any L-edge in α_2 that is not even weakly supported is zero. Let $\ell_1 x \in \alpha_2$ be such an edge. Let $\ell_1\ell_2$ be the middle rail edge and let $\ell_2 y$ be the adjacency in α_2. If $\ell_2 y$ is not weakly supported, $\ell_1\ell_2 yx\ell_1$ is an augmenting cycle. Otherwise, if $\ell_2 y$ is a rail edge, it contributes $+1$ to the score and $\ell_1\ell_2 yx\ell_1$ is a non-negative cycle.

The last case is that $\ell_2 y$ is a rung edge contributing $+1$ to the score. Let $\ell_3 = y$, let $\ell_3\ell_4$ be the other middle rail edge, and let $\ell_4 z$ be the adjacency in α_2. Again, if $\ell_4 z$ is unsupported, $\ell_1\ell_2\ell_3\ell_4 zx\ell_1$ is an augmenting cycle, otherwise it is a rail edge and the cycle is non-negative.

It is easy to check that with each non-negative pair of cycles, we get rid of an L-edge that is not weakly supported, unless we improve the score, which may be done only $O(n)$ times. In the process, we may introduce unsupported C-edges, which is okay and we will deal with them next. □

Proposition 4. *We may suppose that there are no common C-edges other than port edges.*

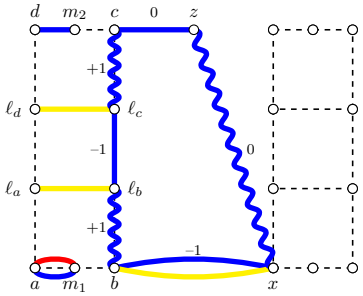
Proof. Let xb be a common C–C-edge in $\alpha_1 \cap \alpha_2$. In the proof, we will use the notation introduced in Fig. 8.5. From what we have proved so far, we may assume that α_1 contains the rung edges $\ell_a\ell_b$ and $\ell_c\ell_d$ (Proposition 1), am_1 is a common adjacency of α_1 and α_2, and either m_2c or m_2d is included in α_2 (Proposition 2).

First, assume the latter case that $m_2d \in \alpha_2$ (Fig. 8.5(a) and 8.5(b)). Since the L-edges are weakly supported, either $\ell_b\ell_c \in \alpha_2$ (Fig. 8.5(a)) or both $\ell_a\ell_b$ and $\ell_c\ell_d$ belong to α_2 (Fig. 8.5(b)). In either case, we can add ladder edges to form an alternating b–c-path with score $+1$ that will be a part of our non-negative pair of cycles.

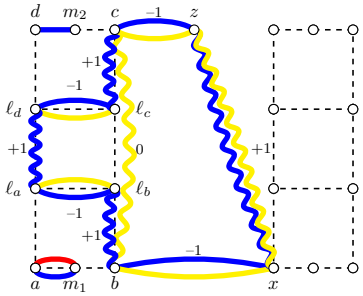
Let cz be an adjacency in α_2. Since m_2 and ℓ_c are already matched to different vertices, cz is unsupported. Now, either $cz \notin \alpha_1$ and $xb\ldots czx$ is a non-negative cycle (see Fig. 8.5(a)), or cz is a common edge and we will also have to exchange some edges in α_1. In particular, $xbczx$ and $xb\ldots czx$ is a non-negative pair of cycles (see Fig. 8.5(b)).

Similarly, we can prove the other case when $m_2c \in \alpha_2$; the non-negative cycle pairs are depicted in Fig. 8.5(c) and 8.5(d). It can be easily checked that the proof also works when extremities x and b belong to the same edge gadget (in this case x coincides with c or d, and b coincides with z). A C–C-edge connecting two corners of a single port is ruled out by Proposition 2.

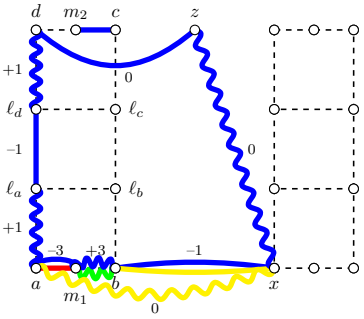
Note that if α_1 and α_2 have a common z-edge (Fig. 8.5(b) and 8.5(d)), we may create a new common unsupported C–C-edge xz. However, the number of common unsupported C–C-edges is decreased by 1 in all cases. □



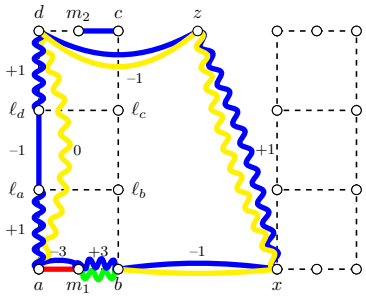
(a) Case 1: α_2 contains m_2d and $cz \notin \alpha_1$.



(b) Case 2: α_2 contains m_2d and $cz \in \alpha_1 \cap \alpha_2$.



(c) Case 3: α_2 contains m_2c and $dz \notin \alpha_1$.



(d) Case 4: α_2 contains m_2c and $dz \in \alpha_1 \cap \alpha_2$.

Figure 8.5: Different cases that arise when disposing of unsupported common C–C-edges. The dashed edges represent the underlying edge gadgets; adjacencies of α_2 are blue, adjacencies of α_1 are yellow, red, and green. Wavy lines are the new suggested adjacencies that should be exchanged for the present ones in the non-negative cycles.

Corollary 3. *We may suppose that all the common adjacencies of the ancestors α_1 and α_2 are weakly supported: $\alpha_1 \cap \alpha_2 \subseteq \Pi$. More specifically, we may suppose that the only common adjacencies are port edges and rung edges. Consequently, each unsupported adjacency except for rung edges in α_2 contributes zero to the score.*

We say that α_1 is *uniform* at a vertex gadget, if all the port edges in the gadget have the same colour (they all agree with either the π_1 edges or the π_2 edges). Next, we prove that α_1 may be assumed uniform at all gadgets. Such an ancestor α_1 directly corresponds to a cut in G.

Here, we use the fact that G is cubic: Imagine that G was a complete bipartite graph $K_{n,n}$ with one more vertex connected to all the other vertices. Then our reduction would not work, since the optimal ancestors would colour one bipartition red, the other green, and the extra vertex half green half red (i.e., half of the ports would be green and the other half red).

First, let us characterize how the non-uniform gadgets look like.

Proposition 5. *We may suppose that the following statements are equivalent:*

- *α_1 is not uniform at a vertex gadget*
- *there is one unsupported I-edge in α_1 incident to the vertex gadget*
- *there is one unsupported C-edge in α_1 incident to the vertex gadget*

Proof. Let α_1 be non-uniform at a vertex gadget. Without loss of generality, let two of the port edges be green and one be red (see Fig. 8.6(a)). Denote r the red and g_1 and g_2 the green edges, such that g_1 is closer to r (as in Fig. 8.6(a)). The edge incident to the intermediate extremity between r and g_1 is an unsupported I-edge.

Obviously, if two *neighbouring* extremities in a vertex gadget are incident with unsupported edges, there is an augmenting cycle, so we may suppose that the intermediate edge between g_1 and g_2 is green and one of the intermediate edges e or f in Fig. 8.6(a) belongs to α_1; the other corner has an unsupported C-edge.

Conversely, if there is an unsupported I-edge or C-edge, the neighbouring ports cannot have edges of the same colour (this would imply two neighbouring extremities with unsupported edges in α_1). □

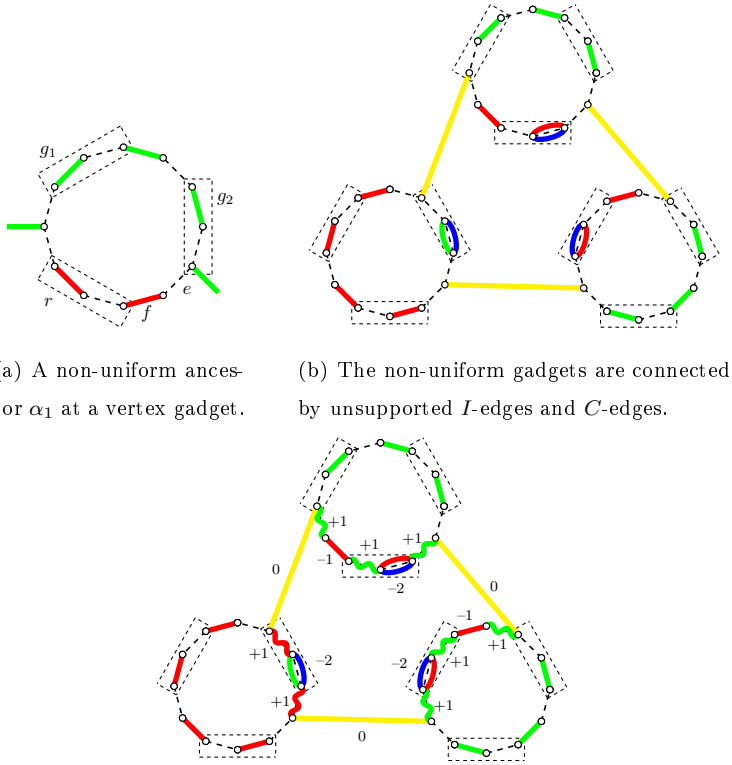
Now we are ready to prove the normal form lemma.

Proposition 6. *We may suppose that in each vertex gadget, the port edges of α_1 are either all red or all green. Thus, we may suppose that all adjacencies in α_1 are supported: $\alpha_1 \subseteq \pi_1 \cup \pi_2$.*

Proof. We prove that for each vertex gadget, we may simply look at the three port edges and choose the colour by majority vote. In the previous proposition, we have shown that non-uniform gadgets have exactly two unsupported edges so they form cycles as in Fig. 8.6(b). Fig. 8.6(c) shows the non-negative cycle that we get by including the edges decided by majority vote. In each vertex gadget, we may lose 1 point for switching the port edge (if this was a common edge), but we get 1 extra point for increasing the number of supported edges. □

Proposition 7. *We may suppose that all adjacencies in α_2 are supported: $\alpha_2 \subseteq \pi_3 \cup \pi_4$.*

Proof. The only remaining unsupported edges in α_2 are the rung edges and C–C-edges. If α_2 contains a rung edge, it must in fact contain both rung edges and in the adjacent ports, only one corner is covered by a port edge. Thus, the edge gadget is incident to two unsupported C–C-edges.



(a) A non-uniform ancestor α_1 at a vertex gadget.

(b) The non-uniform gadgets are connected by unsupported I-edges and C-edges.

(c) A non-negative cycle used for disposing of non-uniform gadgets and unsupported edges in α_1.

Figure 8.6: Non-uniform ancestors at a vertex and a way how to remedy them.

Conversely, it is easy to see that if α_2 contains a C–C-edge, in the incident edge gadgets, α_2 contains either both rung or both middle rail edges and there are C–C-edges incident to the corners of the opposite ports. So the edge gadgets together with the unsupported C–C-edges form cycles and all the rung edges are in these edge gadgets (see Fig. 8.7).

In each edge gadget, we can join the two corners by a non-negative alternating path (see Fig. 8.7); we can lose 1 point for destroying a common adjacency of α_1 and α_2, but we gain 1 point for increasing the number of supported edges in α_2. By exchanging edges along these cycles, we fix both the unsupported C–C-edges and rung edges. □

This concludes the proof of the normal form lemma and thus also the proof of NP-hardness and APX-hardness of the Breakpoint-Quartet problem.

8.5 Conclusion

In this chapter, we have settled several open problems concerning the computational complexity of different rearrangement problems in the breakpoint models. There are at least three intriguing questions

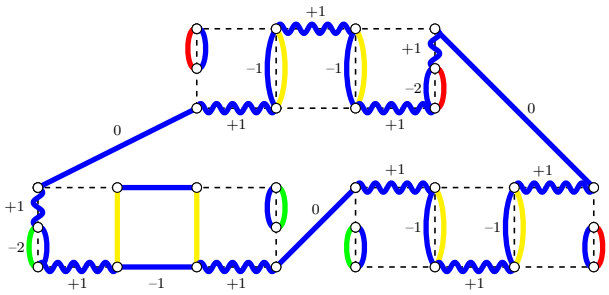


Figure 8.7: Example of three edge gadgets connected in a cycle by unsupported C–C-edges. We can join two corners with unsupported C–C-edges in an edge gadget by a non-negative path. Note that we also get rid of the blue rung edges in the top and right edge gadgets at the same time.

in this area that remain open. The first two are of theoretical interest and are related to approximability of the Small-Phylogeny problem, the third question is more practical:

1. How well can we approximate Small-Phylogeny? For example, Breakpoint-Quartet problem can be easily formulated as an integer linear program (we can use different variables for the edges present only in α_1, only in α_2, and in the intersection $\alpha_1 \cap \alpha_2$). Its relaxation might lead to an algorithm with a good approximation ratio.

2. In the Steinerization approach to ancestral reconstruction, we repeatedly replace the ancestral genomes by medians of genomes in the neighbouring nodes of the tree until we converge to a local optimum. Despite the fact that this is the most common approach to ancestral reconstruction (also in the other models) and that preliminary experiments with simulated data suggest that this heuristic performs very well, no guarantees are known for the method (in any model).

3. Finally, the motivation behind the general breakpoint model is that we can solve the median problem in polynomial time. Using the Steinerization method, we can also get very good solutions of the Small-Phylogeny problem rapidly. The question is: Are these solutions useful in practice? Are they biologically plausible? Or can we adjust them and use them as starting points in more complicated models?

Chapter 9

Conclusion

As we have seen in the previous chapters, much progress has been made in the area of genome rearrangements. Still, many important problems remain open. We conclude with three such problems:

- In Chapter 2, we have seen the great progress on the reversal and RT sorting. The first polynomial algorithm by Hannenhalli and Pevzner has been improved from $O(n^4)$ time to $O(n\sqrt{n})$ time. There is also a practical quadratic algorithm which seems to run in $O(n \log n)$ in average case. It remains an open problem, whether we can sort signed permutations by reversals in $O(n \log n)$ time in worst case.
- There has been considerable progress on the median problem since the first results by Siepel and Caprara. However, our understanding of the slightly more complicated halving problems is lagging behind. The current state of the art median solvers are based on the decomposition theory and adequate subgraphs. Can we extend the theory and apply it to other problems such as double distance, guided halving, genome aliquoting, or even phylogeny problems?
- Finally, a very hard problem, which is widely open: Design a good probabilistic model for genome rearrangements.

Bibliography

Adam, Z. and Sankoff, D. (2008). The ABC of MGR with DCJ. *Bioinformatics*, 4:69–74.

Alekseyev, M. and Pevzner, P. (2007a). Whole genome duplications, multi-break rearrangements, and genome halving problem. In *Proc. SODA*, pages 665–679. Society for Industrial and Applied Mathematics.

Alekseyev, M. A. and Pevzner, P. A. (2007b). Colored de Bruijn graphs and the genome halving problem. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 4(1):98–107.

Alekseyev, M. A. and Pevzner, P. A. (2007c). Whole genome duplications and contracted breakpoint graphs. *SIAM J. Comput.*, 36(6):1748–1763.

Alekseyev, M. A. and Pevzner, P. A. (2008). Multi-break rearrangements and chromosomal evolution. *Theor. Comput. Sci.*, 395(2-3):193–202.

Alimonti, P. and Kann, V. (1997). Hardness of approximating problems on cubic graphs. In *Proc. CIAC*, pages 288–298.

Atteson, K. (1997). The performance of neighbor-joining algorithms of phylogeny reconstruction. In *Proc. COCOON*, pages 101–110. Springer.

Bachrach, A., Chen, K., Harrelson, C., Mihaescu, R., Rao, S., and Shah, A. (2005). Lower bounds for maximum parsimony with gene order data. In *Proc. RECOMB-CG*, pages 1–10. Springer.

Bader, D. A., Moret, B. M. E., and Yan, M. (2001). A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *J. Comput. Biol.*, 8(5):483–491.

Bérard, S., Bergeron, A., Chauve, C., and Paul, C. (2007). Perfect sorting by reversals is not always difficult. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 4(1):4–16.

Bérard, S., Chateau, A., Chauve, C., Paul, C., and Tannier, E. (2009). Computation of perfect DCJ rearrangement scenarios with linear and circular chromosomes. *J. Comput. Biol.*, 16(10):1287–1309.

Bergeron, A. (2005). A very elementary presentation of the Hannenhalli-Pevzner theory. *Discrete Appl. Math.*, 146(2):134–145.

Bergeron, A., Heber, S., and Stoye, J. (2002). Common intervals and sorting by reversals: a marriage of necessity. In *Proc. ECCB*, pages 54–63.

Bergeron, A., Mixtacki, J., and Stoye, J. (2006a). On sorting by translocations. *J. Comput. Biol.*, 13(2):567–578.

Bergeron, A., Mixtacki, J., and Stoye, J. (2006b). A unifying view of genome rearrangements. In *Proc. WABI*, pages 163–173.

Bergeron, A., Mixtacki, J., and Stoye, J. (2008). HP distance via double cut and join distance. In *Proc. CPM*, pages 56–68.

Bergeron, A., Mixtacki, J., and Stoye, J. (2009). A new linear time algorithm to compute the genomic distance via the double cut and join distance. *Theor. Comput. Sci.*, 410(51):5300–5316.

Bergeron, A. and Mixtacki, J. and Stoye, J. (2004). Reversal distance without hurdles and fortresses. In *Proc. CPM*, pages 388–399. Springer.

Berman, P. and Hannenhalli, S. (1996). Fast sorting by reversal. In *Proc. CPM*, pages 168–185.

Berman, P., Hannenhalli, S., and Karpinski, M. (2002). 1.375-approximation algorithm for sorting by reversals. In *Proc. ESA*, pages 401–408. Springer.

Berman, P. and Karpinski, M. (1999). On some tighter inapproximability results (extended abstract). In *Proc. ICALP*, pages 705–705. Springer.

Bernt, M., Merkle, D., and Middendorf, M. (2007). Using median sets for inferring phylogenetic trees. *Bioinformatics*, 23(2):e129.

Biedl, T. C. (2001). Linear reductions of maximum matching. In *Proc. SODA*, pages 825–826.

Bininda-Emonds, O. R. (2004). *Phylogenetic supertrees: combining information to reveal the tree of life*, volume 4. Springer.

Blanchette, M., Bourque, G., and Sankoff, D. (1997). Breakpoint phylogenies. In *Genome Inform. Ser. Workshop Genome Inform.*, pages 25–34.

Blin, G., Chauve, C., and Fertin, G. (2004). The breakpoint distance for signed sequences. In *Proc. CompBioNets*, pages 3–16.

Bourque, G. and Pevzner, P. (2002). Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome Res.*, 12(1):26.

Bourque, G., Zdobnov, E., Bork, P., Pevzner, P., and Tesler, G. (2005). Comparative architectures of mammalian and chicken genomes reveal highly variable rates of genomic rearrangements across different lineages. *Genome Res.*, 15(1):98.

Bruno, W. J., Socci, N. D., and Halpern, A. L. (2000). Weighted neighbor joining: a likelihood-based approach to distance-based phylogeny reconstruction. *Mol. Biol. Evol.*, 17(1):189–197.

Bryant, D. (1998). The complexity of the breakpoint median problem. Technical Report CRM-2579, Centre de Recherches Mathematiques, Universite de Montreal.

Bryant, D. (2000). The complexity of calculating exemplar distances. In Sankoff and Nadeau (2000), pages 207–212.

Bryant, D. (2004). A lower bound for the breakpoint phylogeny problem. *J. Discrete Algorithms*, 2(2):229–255.

Bulteau, L., Fertin, G., and Rusu, I. (2012a). Pancake flipping is hard. In *Proc. MFCS*, pages 247–258. Springer.

Bulteau, L., Fertin, G., and Rusu, I. (2012b). Sorting by transpositions is difficult. *SIAM J. Discrete Math.*, 26(3):1148–1180.

Caprara, A. (1997). Sorting by reversals is difficult. In *Proc. RECOMB*, page 83. ACM.

Caprara, A. (1999). Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM J. Discrete Math.*, 12(1):91–110.

Caprara, A. (2001). On the practical solution of the reversal median problem. In *Proc. WABI*, pages 238–251.

Caprara, A. (2002). Additive bounding, worst-case analysis, and the breakpoint median problem. *SIAM J. Optimiz.*, 13(2):508–519.

Caprara, A. (2003). The reversal median problem. *INFORMS J. Comput.*, 15(1):93.

Chauve, C. and Tannier, E. (2008). A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genomes. *PLoS Comput. Biol.*, 4(11):e1000234.

Chen, Z., Fu, B., and Zhu, B. (2006). The approximability of the exemplar breakpoint distance problem. In *Proc. AAIM*, pages 291–302.

Chitturi, B., Fahle, W., Meng, Z., Morales, L., Shields, C., Sudborough, I., and Voit, W. (2009). An $(18/11)n$ upper bound for sorting by prefix reversals. *Theor. Comput. Sci.*, 410(36):3372–3390.

Chor, B. and Tuller, T. (2005). Maximum likelihood of evolutionary trees: hardness and approximation. *Bioinformatics*, 21(suppl 1):i97–i106.

Chor, B. and Tuller, T. (2006). Finding a maximum likelihood tree is hard. *J. ACM*, 53(5):722–744.

Christie, D. A. (1996). Sorting permutations by block-interchanges. *Inf. Process. Lett.*, 60(4):165–169.

Chrobak, M., Kolman, P., and Sgall, J. (2005). The greedy algorithm for the minimum common string partition problem. *ACM T. Algorithms*, 1(2):350–366.

Chrobak, M., Szymacha, T., and Krawczyk, A. (1990). A data structure useful for finding hamiltonian cycles. *Theor. Comput. Sci.*, 71(3):419–424.

Cohen, D. and Blum, M. (1995). On the problem of sorting burnt pancakes. *Discrete Appl. Math.*, 61(2):105–120.

Cosner, M. (1993). *Phylogenetic and molecular evolutionary studies of chloroplast DNA variation in the Campanulaceae*. PhD thesis, Ohio State University.

Cosner, M., Jansen, R., Moret, B., Raubeson, L., Wang, L., Warnow, T., and Wyman, S. (2000). An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In Sankoff and Nadeau (2000), pages 99–121.

Cui, Y., Wang, L., Zhu, D., and Liu, X. (2008). A $(1.5 + \varepsilon)$-approximation algorithm for unsigned translocation distance. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 5(1):56–66.

Day, W. (1983). Computationally difficult parsimony problems in phylogenetic systematics. *J. Theor. Biol.*, 103(3):429–438.

Dees, J. (2009). *Simultaneous Matchings in Dynamic Graphs*. Student research project, Universität Karlsruhe.

Desper, R. and Gascuel, O. (2004). Theoretical foundation of the balanced minimum evolution method of phylogenetic inference and its relationship to weighted least-squares tree fitting. *Mol. Biol. Evol.*, 21(3):587–598.

Doyon, J.-P., Ranwez, V., Daubin, V., and Berry, V. (2011). Models, algorithms and programs for phylogeny reconciliation. *Brief. Bioinform.*, 12(5):392–400.

El-Mabrouk, N., Bryant, D., and Sankoff, D. (1999). Reconstructing the pre-doubling genome. In *Proc. RECOMB*, pages 154–163. ACM.

El-Mabrouk, N. and Nadeau, J. and Sankoff, D. (1998). Genome halving. In *Proc. CPM*, pages 235–250. Springer.

El-Mabrouk, N. and Sankoff, D. (1999a). Hybridization and genome rearrangement. In *Proc. CPM*, pages 78–87. Springer.

El-Mabrouk, N. and Sankoff, D. (1999b). On the reconstruction of ancient doubled circular genomes using minimum reversals. *Genome Inform. Ser.*, pages 83–93.

El-Mabrouk, N. and Sankoff, D. (2003). The reconstruction of doubled genomes. *SIAM J. Comput.*, 32(3):754–792.

Elias, I. and Hartman, T. (2006). A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 3(4):369–379.

Elias, I. and Tuller, T. (2007). Reconstruction of ancestral genomic sequences using likelihood. *J. Comput. Biol.*, 14(2):216–237.

Erdös, P. L., Soukup, L., and Stoye, J. (2011). Balanced vertices in trees and a simpler algorithm to compute the genomic distance. *Appl. Math. Lett.*, 24(1):82–86.

Eriksen, N. (2002). $(1 + \varepsilon)$-approximation of sorting by reversals and transpositions. *Theor. Comput. Sci.*, 289(1):517–529.

Eriksen, N. (2007). Reversal and transposition medians. *Theor. Comput. Sci.*, 374(1-3):111–126.

Eriksen, N. (2009). Median clouds and a fast transposition median solver. In *Proc. FPSAC*, pages 373–384. DMTCS Proceedings.

Felsenstein, J. (1981). Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, 17(6):368–376.

Felsenstein, J. (2002). *PHYLIP (Phylogeny Inference Package) version 3.6 a3.*

Feng, J. and Zhu, D. (2007). Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM T. Algorithms*, 3(3).

Fertin, G., Labarre, A., and Rusu, I. (2009). *Combinatorics of genome rearrangements.* The MIT Press.

Figeac, M. and Varré, J.-S. (2004). Sorting by reversals with common intervals. In *Proc. WABI*, pages 26–37.

Fitch, W. and Margoliash, E. (1967). Construction of phylogenetic trees. *Science*, 155(760):279–284.

Foulds, L. and Graham, R. (1982). The Steiner problem in phylogeny is NP-complete. *Adv. Appl. Math.*, 3(1):43–49.

Gabow, H. (1973). *Implementation of algorithms for maximum matching on nonbipartite graphs.* PhD thesis, Stanford University.

Gabow, H. (1990). Data structures for weighted matching and nearest common ancestors with linking. In *Proc. SODA*, pages 434–443. Society for Industrial and Applied Mathematics.

Gabow, H. and Tarjan, R. (1991). Faster scaling algorithms for general graph matching problems. *J. ACM*, 38(4):815–853.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.

Gavranović, H., Chauve, C., Salse, J., and Tannier, E. (2011). Mapping ancestral genomes with massive gene loss: A matrix sandwich problem. *Bioinformatics*, 27(13):i257–i265.

Gavranović, H. and Tannier, E. (2010). Guided genome halving: provably optimal solutions provide good insights into the preduplication ancestral genome of *Saccharomyces cerevisiae*. In *Proc. Pac. Symp. Biocomp.*, volume 15, pages 21–30.

Goldstein, A., Kolman, P., and Zheng, J. (2005). Minimum common string partition problem: Hardness and approximations. *Electr. J. Comb.*, 12.

Guindon, S., Dufayard, J.-F., Lefort, V., Anisimova, M., Hordijk, W., and Gascuel, O. (2010). New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of phyml 3.0. *Syst. Biol.*, 59(3):307–321.

Guindon, S. and Gascuel, O. (2003). A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst. Biol.*, 52(5):696–704.

Han, Y. (2006). Improving the efficiency of sorting by reversals. In Arabnia, H. R. and Valafar, H., editors, *Proc. Pac. Symp. Biocomp.*, pages 406–409. CSREA Press.

Hannenhalli, S. and Pevzner, P. (1999). Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *J. ACM*, 46(1):1–27.

Hannenhalli, S. and Pevzner, P. A. (1995). Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proc. FOCS*, pages 581–592.

Hasegawa, M., Kishino, H., and Yano, T. (1985). Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *J. Mol. Evol.*, 22(2):160–174.

Heydari, M. H. and Sudborough, I. H. (1997). On the diameter of the pancake network. *J. Algorithm.*, 25(1):67–94.

Jackson, B., Schnable, P., and Aluru, S. (2007). Consensus genetic maps as median orders from inconsistent sources. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, pages 161–171.

Jean, G. and Nikolski, M. (2007). Genome rearrangements: a correct algorithm for optimal capping. *Inf. Process. Lett.*, 104(1):14–20.

Jukes, T. and Cantor, C. (1969). Evolution of protein molecules. *Mammalian protein metabolism*, 3:121–132.

Kaplan, H., Shamir, R., and Tarjan, R. E. (1999). A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J. Comput.*, 29(3):880–892.

Kaplan, H. and Verbin, E. (2005). Sorting signed permutations by reversals, revisited. *J. Comput. Syst. Sci.*, 70(3):321–341.

Kimura, M. (1980). A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.*, 16(2):111–120.

Knuth, D. (1973). *The art of computer programming, Vol. 3.* Addison-Wesley, Reading, MA.

Kolman, P. and Walen, T. (2007). Reversal distance for strings with duplicates: Linear time approximation using hitting set. *Electr. J. Comb.*, 14(1).

Kováč, J., Braga, M. D. V., and Stoye, J. (2010). The problem of chromosome reincorporation in DCJ sorting and halving. In *Proc. RECOMB-CG*, pages 13–24.

Kováč, J., Brejová, B., and Vinař, T. (2011a). A practical algorithm for ancestral rearrangement reconstruction. In Przytycka and Sagot (2011), pages 163–174.

Kováč, J., Warren, R., Braga, M. D. V., and Stoye, J. (2011b). Restricted DCJ model: Rearrangement problems with chromosome reincorporation. *J. Comput. Biol.*, 18(9):1231–1241.

Lander, E., Linton, L., Birren, B., Nusbaum, C., Zody, M., Baldwin, J. and Devon, K., Dewar, K., Doyle, M., FitzHugh, W., et al. (2001). Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921.

Larget, B., Kadane, J., and Simon, D. (2005). A Bayesian approach to the estimation of ancestral genome arrangements. *Mol. Phylogenet. Evol.*, 36(2):214–223.

Lawler, E. (1976). *Combinatorial optimization: networks and matroids.* Holt, Rinehart and Winston.

Lin, Y. and Moret, B. M. (2008). Estimating true evolutionary distances under the dcj model. *Bioinformatics*, 24(13):i114–i122.

Lin, Y., Rajan, V., Swenson, K. M., and Moret, B. M. (2010). Estimating true evolutionary distances under rearrangements, duplications, and losses. *BMC Bioinformatics*, 11(Suppl 1):S54.

Ma, J. and Zhang, L., Suh, B., Raney, B.J. and Burhans, R., Kent, W.J. and Blanchette, M., Haussler, D., and Miller, W. (2006). Reconstructing contiguous regions of an ancestral genome. *Genome Res.*, 16(12):1557.

Maddison, D. and Maddison, W. (2000). *MacClade 4.0: Analysis of phylogeny and character evolution.* Sinauer Associates, Sunderland, Mass.

Maddison, W. and Maddison, D. (2004). Mesquite: a modular system for evolutionary analysis.

Maddison, W. P. (1997). Gene trees in species trees. *Syst. Biol.*, 46(3):523–536.

Micali, S. and Vazirani, V. V. (1980). An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *Proc. FOCS*, pages 17–27. IEEE Computer Society.

Mixtacki, J. (2008). Genome halving under DCJ revisited. *Computing and Combinatorics*, pages 276–286.

Moret, B., Tang, J. and Wang, L., and Warnow, T. (2002a). Steps toward accurate reconstructions of phylogenies from gene-order data. *J. Comput. Syst. Sci.*, 65(3):508–525.

Moret, B. and Tang, J. and Warnow, T. (2005). Reconstructing phylogenies from gene-content and gene-order data. *Mathematics of Evolution and Phylogeny*, pages 321–352.

Moret, B., Wang, L., Warnow, T., and Wyman, S. (2001a). New approaches for reconstructing phylogenies from gene order data. *Bioinformatics*, 17(S1):S165.

Moret, B. M. (2005). Computational challenges from the tree of life. In *Proc. ALENEX*, pages 3–16. SIAM.

Moret, B. M. E., Siepel, A. C., Tang, J., and Liu, T. (2002b). Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In *Proc. WABI*, pages 521–536.

Moret, B. M. E., Wyman, S. K., Bader, D. A., Warnow, T., and Yan, M. (2001b). A new implementation and detailed study of breakpoint analysis. In *Proc. Pac. Symp. Biocomp.*, pages 583–594.

Nei, M. and Kumar, S. (2000). *Molecular evolution and phylogenetics*. Oxford University Press, USA.

Ozery-Flato, M. and Shamir, R. (2003). Two notes on genome rearrangement. *J. Bioinform. Comput. Biol.*, 1(1):71–94.

Ozery-Flato, M. and Shamir, R. (2006). An $O(n^{3/2}\sqrt{\log n})$ algorithm for sorting by reciprocal translocations. In Lewenstein, M. and Valiente, G., editors, *Proc. CPM*, volume 4009 of *LNCS*, pages 258–269. Springer.

Ozery-Flato, M. and Shamir, R. (2007). Rearrangements in genomes with centromeres part i: Translocations. In *Proc. RECOMB*, pages 339–353.

Page, R. D. and Charleston, M. A. (1997). From gene to organismal phylogeny: reconciled trees and the gene tree/species tree problem. *Mol. Phylogenet. Evol.*, 7(2):231–240.

Pe'er, I. and Shamir, R. (1998). The median problems for breakpoints are NP-complete. *Electronic Colloquium on Computational Complexity (ECCC)*, 5(71).

Pe'er, I. and Shamir, R. (2000). Approximation algorithms for the median problem in the breakpoint model. In Sankoff and Nadeau (2000), pages 225–241.

Plesník, J. (1979). The NP-completeness of the hamiltonian cycle problem in planar digraphs with degree bound two. *Inf. Process. Lett.*, 8(4):199–201.

Przytycka, T. M. and Sagot, M.-F., editors (2011). *Proc. WABI*, volume 6833 of *Lect. Notes in Computer Sci.* Springer.

Radcliffe, A., Scott, A., and Wilmer, E. (2006). Reversals and transpositions over finite alphabets. *SIAM J. Discrete Math.*, 19(1):224.

Rajan, V., Xu, A. W., Lin, Y., Swenson, K. M., and Moret, B. M. E. (2010). Heuristics for the inversion median problem. *BMC Bioinformatics*, 11(S-1):30.

Rokas, A. and Holland, P. W. (2000). Rare genomic changes as a tool for phylogenetics. *Trends Ecol. Evol.*, 15(11):454–459.

Ronquist, F. and Huelsenbeck, J. (2003). MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572.

Sankoff, D. (1992). Edit distances for genome comparisons based on non-local operations. In *Proc. CPM*, pages 121–135.

Sankoff, D. and Blanchette, M. (1997). The median problem for breakpoints in comparative genomics. In *Proc. COCOON*, pages 251–264.

Sankoff, D. and Blanchette, M. (1998). Multiple genome rearrangement and breakpoint phylogeny. *J. Comput. Biol.*, 5(3):555–570.

Sankoff, D., Bryant, D., Deneault, M., Lang, B., and Burger, G. (2000). Early eukaryote evolution based on mitochondrial gene order breakpoints. *J. Comput. Biol.*, 7(3-4):521–535.

Sankoff, D., Cedergren, R. J., and Lapalme, G. (1976). Frequency of insertion-deletion, transversion, and transition in the evolution of 5s ribosomal rna. *J. Mol. Evol.*, 7(2):133–149.

Sankoff, D. and Nadeau, J. H., editors (2000). *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, volume 1 of *Computational Biology Series*. Kluwer Academic Publishers.

Sankoff, D., Sundaram, G., and Kececioglu, J. D. (1996). Steiner points in the space of genome rearrangements. *Int. J. Found. Comput. Sci.*, 7(1):1–9.

Seoighe, C. and Wolfe, K. H. (1998). Extent of genomic rearrangement after genome duplication in yeast. *Proc. Natl. Acad. Sci.*, 95(8):4447–4452.

Siepel, A. C. and Moret, B. M. E. (2001). Finding an optimal inversion median: Experimental results. In *Proc. WABI*, pages 189–203.

Simmonds, N. W. et al. (1976). *Evolution of crop plants*. Longman Group Ltd.

Stamatakis, A. (2006). Raxml-vi-hpc: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690.

Stamatakis, A., Ludwig, T., and Meier, H. (2005). RAxML-III: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21(4):456–463.

Studier, J. A., Keppler, K. J., et al. (1988). A note on the neighbor-joining algorithm of saitou and nei. *Mol. Biol. Evol.*, 5(6):729–731.

Swenson, K. M., Badr, G., and Sankoff, D. (2011). Listing all sorting reversals in quadratic time. *Algorithm. Mol. Biol.*, 6:11.

Swenson, K. M. and Moret, B. M. E. (2009). Inversion-based genomic signatures. *BMC Bioinformatics*, 10(S-1).

Swenson, K. M., Rajan, V., Lin, Y., and Moret, B. M. E. (2010). Sorting signed permutations by inversions in $O(n \log n)$ time. *J. Comput. Biol.*, 17(3):489–501.

Swofford, D. (2003). *PAUP*. Phylogenetic analysis using parsimony (*and other methods). Version 4.*

Tamura, K. (1992). Estimation of the number of nucleotide substitutions when there are strong transition-transversion and G+ C-content biases. *Mol. Biol. Evol.*, 9(4):678.

Tamura, K. and Nei, M. (1993). Estimation of the number of nucleotide substitutions in the control region of mitochondrial DNA in humans and chimpanzees. *Mol. Biol. Evol.*, 10(3):512.

Tamura, K., Peterson, D., Peterson, N., Stecher, G., Nei, M., and Kumar, S. (2011). MEGA5: Molecular evolutionary genetics analysis using maximum likelihood, evolutionary distance, and maximum parsimony methods. *Mol. Biol. Evol.*, 28(10):2731–2739.

Tang, J. and Moret, B. (2005). Linear programming for phylogenetic reconstruction based on gene rearrangements. In *Proc. CPM*, pages 406–416. Springer.

Tang, J. and Wang, L.-S. (2005). Improving genome rearrangement phylogeny using sequence-style parsimony. In *Proc. BIBE*, pages 137–144. IEEE.

Tannier, E., Bergeron, A., and Sagot, M.-F. (2007). Advances on sorting by reversals. *Discrete Appl. Math.*, 155(6-7):881–888.

Tannier, E., Zheng, C., and Sankoff, D. (2009). Multichromosomal median and halving problems under different genomic distances. *BMC Bioinformatics*, 10.

Tesler, G. (2002). Efficient algorithms for multichromosomal genome rearrangements. *J. Comput. Syst. Sci.*, 65(3):587–609.

Valach, M., Farkaš, Z., Fričová, D., Kováč, J., Brejová, B., Vinař, T., Pfeiffer, I., Kucsera, J., Tomáška, Ľ., Lang, B. F., and Nosek, J. (2011). Evolution of linear chromosomes and multipartite genomes in yeast mitochondria. *Nucleic Acids Res.*, 39(10):4202–4219.

Wang, L.-S., Jansen, R. K., Moret, B. M. E., Raubeson, L. A., and Warnow, T. (2002). Fast phylogenetic methods for the analysis of genome rearrangement data: An empirical study. In *Proc. Pac. Symp. Biocomp.*, pages 524–535.

Wang, L.-S. and Warnow, T. (2001). Estimating true evolutionary distances between genomes. In *Proc. STOC*, pages 637–646. ACM.

Warnow, T. (2006). Disk covering methods: Improving the accuracy and speed of large-scale phylogenetic analyses. In *Handbook of computational molecular biology*, volume 9. CRC Press.

Warren, R. and Sankoff, D. (2009a). Genome aliquoting with double cut and join. *BMC Bioinformatics*, 10(Suppl 1):S2.

Warren, R. and Sankoff, D. (2009b). Genome halving with double cut and join. *J. Bioinform. Comput. Biol.*, 7(2):357–371.

Warren, R. and Sankoff, D. (2011). Genome aliquoting revisited. *J. Comput. Biol.*, 18(9):1065–1075.

Wheeler, T. J. (2009). Large-scale neighbor-joining with NINJA. In Salzberg, S. and Warnow, T., editors, *Proc. WABI*, volume 5724 of *Lect. Notes in Computer Sci.*, pages 375–389. Springer.

Xu, A. W. (2009). DCJ median problems on linear multichromosomal genomes: Graph representation and fast exact solutions. In Ciccarelli, F. and Miklós, I., editors, *Proc. RECOMB-CG*, volume 5817 of *Lect. Notes in Computer Sci.*, pages 70–83. Springer.

Xu, A. W. and Moret, B. M. E. (2011). GASTS: parsimony scoring under rearrangements. In Przytycka and Sagot (2011), pages 351–363.

Xu, A. W. and Sankoff, D. (2008). Decompositions of multiple breakpoint graphs and rapid exact solutions to the median problem. In *Proc. WABI*, pages 25–37.

Yancopoulos, S., Attie, O., and Friedberg, R. (2005). Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346.

Zhang, M., Arndt, W., and Tang, J. (2008). A branch-and-bound method for the multichromosomal reversal median problem. In Crandall, K. A. and Lagergren, J., editors, *Proc. WABI*, volume 5251 of *Lect. Notes in Computer Sci.*, pages 14–24. Springer.

Zhang, M., Arndt, W., and Tang, J. (2009). An exact solver for the DCJ median problem. In Altman, R. B., Dunker, A. K., Hunter, L., Murray, T., and Klein, T. E., editors, *Proc. Pac. Symp. Biocomp.*, pages 138–149.

Zheng, C., Zhu, Q., Adam, Z., and Sankoff, D. (2008). Guided genome halving: hardness, heuristics and the history of the Hemiascomycetes. In *Proc. ISMB*, pages 96–104.

Zheng, C., Zhu, Q., and Sankoff, D. (2006). Genome halving with an outgroup. *Evol. Bioinform. Online*, 2:295.

Zhu, D. and Wang, L. (2006). On the complexity of unsigned translocation distance. *Theor. Comput. Sci.*, 352(1-3):322–328.