Rearrangement problems with duplicated genomic markers

Antoine Thomas

Problèmes de réarrangement avec marqueurs génomiques dupliqués

Rearrangement Problems with duplicated genomic content

THÈSE

présentée et soutenue publiquement le 18 juillet 2014

pour l'obtention du

Doctorat de l'Université de Lille 1 – Sciences et Technologies

(spécialité informatique)

par

Antoine Thomas

Composition du jury

Rapporteurs :	David Sankoff, Canada Research Chair	University of Ottawa, ON, Canada
	Sophia Yancopoulos, Science Writer, *co-rapporteur*	Feinstein Inst. for Med. Research, USA
	Eric Tannier, CR Inria	INRIA Grenoble, LBBE, Univ. Lyon 1
Examinateurs :	Joachim Niehren, DR Inria, *président du jury*	INRIA Lille, LIFL, Univ. Lille 1
	François Boulier, Professeur, *directeur de thèse*	LIFL, Univ. Lille 1

UNIVERSITÉ DE LILLE 1 – SCIENCES ET TECHNOLOGIES
ÉCOLE DOCTORALE SCIENCES POUR L'INGÉNIEUR
Laboratoire d'Informatique Fondamentale de Lille — UMR 8022
U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : +33 (0)3 28 77 85 41 – Télécopie : +33 (0)3 28 77 85 37 – email : direction@lifl.fr

Contents

Avant-propos et remerciements

Cette thèse de doctorat présente le travail que j'ai fourni de 2010 à fin 2012 au sein de l'équipe bonsai du LIFL.

Les lecteurs les plus attentifs remarqueront que la soutenance n'a pourtant pas eu lieu avant juillet 2014.

J'ai en effet été contraint d'interrompre mes activités de recherche au profit d'une activité bien moins amusante, une lutte contre un harcèlement perpétré impunément par les membres les plus haut placés de cette équipe de recherche. Dès lors que j'ai eu le malheur de dire non à leur appropriation honteuse de mon travail et à leur manque d'éthique en général, j'ai subi de nombreuses attaques dénigrant non seulement mon travail mais aussi ma personne au sein de toute l'équipe.

Le moins que je puisse dire c'est qu'il est malheureusement très difficile, en tant que simple doctorant, de défendre la valeur de son travail et de s'opposer à ces abus, dans un système où les médiateurs et supérieurs hiérarchiques ne sont autre que les collègues voire amis de nos bourreaux, et où nos pairs sont bien trop lâches pour faire preuve de solidarité.

Il n'est donc pas étonnant dans ce contexte, le sentiment d'impunité aidant, de voir la banalisation des abus de pouvoir au détriment de toute déontologie. A croire que pour certains il est très difficile de comprendre que *docteur* ne signifie pas *médecin*, et qu'ils n'ont donc aucune légitimité quand il s'agit d'établir des diagnostics d'ordre psychiatrique à l'égard de qui que ce soit...

Par ailleurs, discuter des problèmes médicaux de mes collègues, d'anciens membres de l'équipe ou de leurs entourages ne fait pas non plus partie de leurs fonctions et ils n'ont donc pas le droit d'en faire étalage public sans le consentement des intéressés, mais j'imagine que le concept de secret professionnel, ce doit être encore plus difficile à comprendre...

⋆ ⋆ ⋆

Tout d'abord merci à Mr Olivier Colot, directeur de l'école doctorale ED SPI 072, d'avoir été l'acteur principal de la résolution du problème en me permettant d'entamer une démarche de changement de directeur de thèse et d'équipe de recherche.

Merci à Mr Michel Petitot pour son temps et ses discussions variées toujours intéressantes, qui prennent le plus souvent une dimension très philosophique. Merci à lui d'incarner ce que devrait être selon moi un chercheur, une personne en éternelle quête de connaissances.

Merci aux examinateurs de cette thèse, Eric Tannier et Sophia Yancopoulos pour leur présence lors de la soutenance et leurs commentaires pertinents concernant ce travail. Cela peut sembler être peu de choses, mais croyez-moi, quand on sort de plus d'un an et demi de critiques infondées et d'injures, de combat contre des gens qui dénigrent ce travail pour mieux se l'approprier ensuite (par exemple en prétendant qu'ils ont été là pour le corriger et que tout le mérite leur revient donc), et qu'on tombe ensuite sur de vrais chercheurs, le contraste est pour le moins saisissant.

En somme, merci à tous ceux qui ont contribué au bon déroulement de cette fin de thèse, portant ainsi sur leurs épaules, bien malgré eux, le poids de l'incompétence de mes anciens encadrants.

Quant à ces derniers, parce que j'ai l'honnêteté qu'ils n'ont pas, je les remercie tout de même pour leur pathétique contribution, qui se résume très précisément à trois points : me donner quelques problèmes sur lesquels travailler par moi-même pendant qu'ils demandent cupidement de nouveaux résultats, tracer quelques figures pour mes articles, et bien-sûr y reformuler des paragraphes (il semblerait que cela soit une de leurs spécialités).

Mes remerciements les plus importants vont naturellement à ma famille, pour m'avoir inculqué des valeurs apparemment rares telles que l'intégrité, le courage et la persévérance, ainsi qu'un certain sens de la justice.

Merci également à mes amis d'avoir été là, c'est important de pouvoir rire en toutes circonstances, et aussi d'avoir du soutien.

Un non-merci à tous ceux qui m'ont proposé, avec le sourire, d'accepter ces injustices, et de cautionner ma propre exploitation, en prétendant que c'était pour mon bien. Ils se reconnaîtront.

Un non-merci à tous ceux qui contribuent au vol de propriété intellectuelle organisé, à cette imposture qu'est la recherche dans certaines équipes où mélange des genres, harcèlement et impunité font un ménage à trois des plus malsains.

Introduction

It is commonly known that life on earth is made of DNA, which carries genetic information, and that it evolves time after time as generations succeed each other.

DNA is a nucleic acid constituted by 4 nucleobases (its building blocks), which are guanine, adenine, thymine, and cytosine, respectively written G, A, T and C.

These nucleotides are folded in a double-helix structure where each base type is combined with its complementary base. G goes with C, A goes with T.

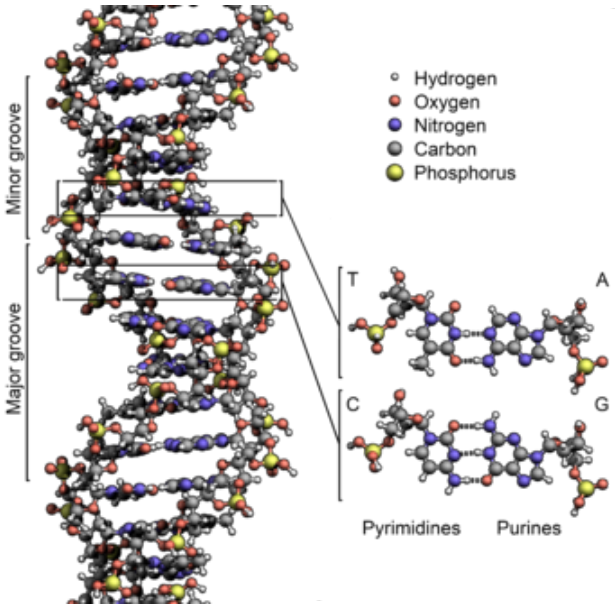


Figure 1: The DNA molecule structure. Image courtesy of http://en.wikipedia.org/wiki/DNA

Biologists observed that evolution also occurred at large scale with genes or group of genes recombinations such as reversal of segments [Sturtevant, 1921] [Palmer and Herbon, 1988].

Understanding rearrangement dynamics is a major issue in *phylogenetics*, the study of evolutionary relationships between species or populations.

One aspect of phylogenetics is trying to reconstruct evolutionary trees. In order to reach this goal, naturally it would be very helpful to be able to determine a relative *evolutionary distance* between species (ie. knowing which species are closer with respect to which others), or to be able, given a group of genomes, to reconstruct the genome of a closest common ancestor to the group.

This is where genome rearrangements come into play. By defining a genome representation

and a set of allowed operations (manipulations), one should be able to define a *distance* (in the mathematical sense, see the following box if needed) between genomes.

A distance function is a function defined on a set X, which for any couple of elements from the set associates a real number, and satisfies 4 simple properties.

The distance d is a function defined as:

$d : X \times X \to R$

and $\forall x, y, z \in X$, we have:

1. $d(x, y) \geq 0$ (a distance cannot be negative)
2. $d(x, y) = 0 \iff x = y$ (distance 0 means the compared elements are equal)
3. $d(x, y) = d(y, x)$ (symmetry)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

A rearrangement problem is always defined by a *starting genome*, a *goal genome*, and a set of allowed operations. Of course, since DNA sequences are a huge amount of data, algorithmic complexity of rearrangement problems is a major point of interest.

While DNA can ultimately be represented by the sequence of its nucleotides, comparative genomics usually deals with genomes at the scale of *genomic markers* (ie. genes or group of genes). Each marker is labeled by an integer, with the sign representing its orientation on the genome.

Here is an example of rearrangement scenario between genomes $A = (\circ\ \ 1\ \ 2\ \ -5\ \ -7\ \ -6\ \ 3\ \ 4\ \ 8\ \ 9\ \ 10\ \ \circ)$ and $B = (\circ\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ 9\ \ 10\ \ \circ)$, with *reversals* as the only allowed operations.

$$(\circ\ 1\ 2\ -5\ \underline{-7\ -6\ 3\ 4}\ 8\ 9\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ \underline{-5\ -4\ -3}\ 6\ 7\ 8\ 9\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \circ)$$

In this example, two reversals suffice to go from the starting genome to the goal, and it cannot be done in one operation. Thus we say that the *reversal distance* between genomes A and B is equal to 2, while the example itself is an *optimal scenario.*

Computing the distance and computing an optimal scenario are two linked but different problems, with varying complexity.

Rearrangement problems were first introduced, for the reversal model, in [Sturtevant, 1921] [Sturtevant and Novitski, 1941] and later rediscovered in [Palmer and Herbon, 1988]. For more explanation, István Miklós wrote a detailed history of genome rearrangements, whose reading is much recommended[1]. Another much recommended reading would be the book "*Combinatorics of Genome Rearrangement*" published by MIT Press, as it presents a mathematically oriented review of the field.

Rearrangement problems were first defined on *non-duplicated* genomes, meaning each gene appears only once in both genomes, which made rearrangement problems in fact permutation sorting problems.

Among the pioneer results of the field there is a polynomial solution for sorting permutations by reversals [Hannenhalli and Pevzner, 1995a], then generalized into the *genomic distance*, a mix of reversals and translocations [Hannenhalli and Pevzner, 1995b]. The DCJ (double-cut and join), a further generalization of operation models was introduced in [Yancopoulos et al., 2005] which also allowed a better framework to study previous operation models.

Generally, it appears that rearrangement problems on non-duplicated genomes usually have polynomial complexity while duplicated markers induce NP-hardness. However, there are exceptions as the definition of a particular operation model or that of a goal genome might imply additional restrictions or release constraints, which can alter complexity.

For example, the *genome halving* problem [El-Mabrouk and Sankoff, 2003][Mixtacki, 2008] is a polynomial problem on duplicated genomes, which released the constraint of a particular goal genome, asking for a more general kind of configuration instead.

It is to note that while the distance and scenario are the usual questions when it comes to rearrangement problems, a substantious amount of work has been made to solve the question of solution spaces (the set of all optimal solutions) of rearrangement problems [Braga et al., 2007] [Braga, 2009] [Braga and Stoye, 2010].

During the preparation of this Ph.D. thesis, I have been working on genome rearrangement problems with duplicated markers. Following the genome halving example, I studied *other* hypotheses that could account for the presence of duplicated markers in genomes. I designed several rearrangement problems to account for these hypotheses and settle their algorithmic complexity.

I proved that under the hypothesis a single tandem duplication event is responsible for all observed duplicated markers, a parcimonious non-duplicated ancestor genome could be inferred in polynomial time. I provided $O(n^2)$ algorithms for the scenario, and a $O(n)$ computation for the distance, for two operation models, namely DCJ and Block Interchange.

I developed several problems based on multiple tandem reconstruction, and proved their NP-hardness.

I also studied breakpoint duplication, an intermediate model where any operation could lead to the duplication of markers at its endpoints. I proved NP-hardness of this problem and designed a fixed-parameter tractable (FPT) algorithm in the number of cycles present in the graph.

This work led to several publications [Thomas et al., 2011] [Thomas et al., 2012a] [Thomas et al., 2012b] [Thomas et al., 2013].

[1] http://www.renyi.hu/~miklosi/AlgorithmsOfBioinformatics.pdf

⋆ ⋆ ⋆

In the first chapter I will informally introduce what a rearrangement problem is, by revisiting a simple problem, *Sorting by Block Interchange*, first solved in [Christie, 1996].

I will then present a literature review, to provide a backdrop for my work, but also to give the reader a first intuition on what is hard and what is not when it comes to rearrangement problems.

A presentation of my own work will follow, sorted by duplication model and operation model, then a general conclusion will close the document.

Chapter 1

Preliminary game: sorting by block interchange

Loosely based on my experience explaining what is my research to my laymen friends and family members, this rather informal chapter is meant as a playful introduction, whose purpose is to give the reader a rough understanding of the basic workings of rearrangement problems and my way of tackling them, even if they are not familiar at all with the field.

I will revisit "Sorting permutations by Block Interchange" [Christie, 1996], then quickly explain how it relates to genome rearrangement and my own work.

1.1 Rules

Let's assume we have a sequence of numbers we want to sort into numerical order.

$$(\circ\ 4\ 5\ 3\ 2\ 10\ 7\ 8\ 1\ 9\ 6\ \circ)$$

The first thing we have to do is define how we are allowed to alter the sequence in order to sort it.

For example, selecting two segments and swapping them is one way to alter a sequence. This operation is self-explanatorily called a *block interchange* (BI). Is it possible to sort the sequence using only this operation?

$$(\circ\ \boxed{4\ 5\ 3\ 2\ 10}\ 7\ 8\ \boxed{1\ 9\ 6}\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ \boxed{9}\ 6\ 7\ 8\ 4\ 5\ 3\ \boxed{2}\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ \boxed{6\ 7\ 8}\ 4\ 5\ \boxed{3}\ 9\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \circ)$$

Yes, it is possible to sort the sequence that way. Was it optimal? Yes, it took only three steps and it's not possible to sort it in two or less.

How do we know that? This is where it gets interesting.

1.2 Breakpoints

For better understanding how sorting works, let's review the very last BI operation, the one that restored the complete numerical order.

$$(\circ\ 1\ 2\ \boxed{6\ 7\ 8}\ 4\ 5\ \boxed{3}\ 9\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \circ)$$

If given the task to sort such sequence using only one BI, anybody would correctly find the solution.

Anybody will instinctively notice that when selecting blocks, all starting/ending points are not equal (for example it would not make sense to cut between 1 and 2 given they are already in their correct relative order).

In fact it only makes sense to cut between numbers that *break* the desired order. Those positions will be called *breakpoints*. Note that when the sequence is fully sorted, no breakpoint remains.

Here is the sequence again, with breakpoints indicated by black triangles.

$$(\circ\ 1\ 2\ \blacktriangle\ 6\ 7\ 8\ \blacktriangle\ 4\ 5\ \blacktriangle\ 3\ \blacktriangle\ 9\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \circ)$$

And of course those black triangles coincide with the extremities of the blocks for the sorting operation.

A breakpoint will never vanish by itself, so if a BI operation can act on at most 4 breakpoints at once, then in the best case, it will manage to solve the four of them and leave the rest of the sequence untouched.

1.3 Example review

With that new information in mind, if we have another look at the starting sequence, we can see that it contains 9 breakpoints.

$$(\circ\ \blacktriangle\ 4\ 5\ \blacktriangle\ 3\ \blacktriangle\ 2\ \blacktriangle\ 10\ \blacktriangle\ 7\ 8\ \blacktriangle\ 1\ \blacktriangle\ 9\ \blacktriangle\ 6\ \blacktriangle\ \circ)$$

Since a BI can only solve at most 4 of them at once, it is now obvious that it is not possible to sort the sequence in less than 3 steps.

We just solved the problem for *this particular sequence*, but we want to be able to give an optimal solution for any sequence, therefore we cannot consider the problem is solved yet.

1.4 General case

When reviewing the full transformation sequence, you might notice that the first BI solved 3 breakpoints ($_{\blacktriangle}1$, $6_{\blacktriangle}$ and $10_{\blacktriangle}$), the second solved 2 ($_{\blacktriangle}2$ and $9_{\blacktriangle}$), and only the last one solved the remaining 4 ($_{\blacktriangle}6$, $8_{\blacktriangle}$, $_{\blacktriangle}3$ and $3_{\blacktriangle}$).

It is easy to solve any 2 given breakpoints with a BI, but solving a 3rd and a 4th will happen only under certain conditions, and by a quick glance you can already tell that the special conditions are not obvious...

The real difficulty we are faced is to design a way to be able to understand those conditions and predict the best course of action, ie. being able of computing the minimum number of operations for **any** given sequence.

While the sequence embeds information about the number of operations needed to sort it, it embeds it in an implicit way. By changing the way we represent it, we might make the operations more explicit.

1.4.1 Drawing a graph I

We established earlier that breakpoints are crucial elements when it comes to sorting, and noted that what characterizes a sorted sequence is *the absence of breakpoints*.

Our goal is to make things more explicit, therefore we will design a data structure around the sequence (which really is another way of representing the same information contained in a sequence), focused on that absence of breakpoints.

To achieve this I will draw a dot between numbers that are in consecutive order (see figure 1.1.

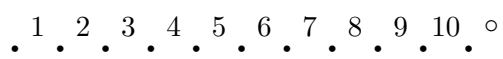


Figure 1.1: Graph for a sorted sequence

Note that I also put dots before the number 1, and after the number 10. Since they are the ends of our sequence, they must be consecutive to the corresponding ∘ symbols.

1.4.2 Drawing a graph II

We designed a data structure around a sorted sequence, and will now generalize it to our shuffled sequence.

First the good cases: “after 4 there is 5” “after 7 there is 8”

Therefore we can add a dot between 4 and 5, as well as another one between 7 and 8 (see figure 1.2). These dots show the parts of the sequence that are in correct order.

So far it is not conceptually different from counting the breakpoints (we are counting *adjacencies* instead), but in order to get more information we need something to do with the rest of the slots, where it is not possible to put dots.

Since it’s about understanding the degree of shuffling, we will think in terms of what we *should* have instead of what we actually have in the sequence.

"After 5 there should be 6".

So let's put a dot after 5 and join it with another dot before 6, to indicate they should be consecutive.

That reasoning alone would allow us to fill in the blanks (follow along on figure 1.2, as horizontal segments A, B and C are drawn from top to bottom):

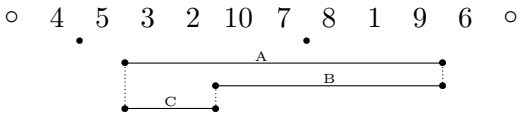


Figure 1.2: Two 1-cycles and one 3-cycle so far.

After 5 → Before 6

We draw a segment joining a dot after 5 with another one before 6 (segment A).

In the sequence, we have 9 6 so by arriving before 6 we are also in the position after 9. We have to put another dot there, accounting for the number 9.

After 9 → Before 10

The dot after 9 is joined with another one put before 10 (segment B), and thus we end up after 2.

After 2 → Before 3

We draw segment C and we are back to our starting position in 3 steps. We completed a *cycle* of length 3 as seen in figure 1.2. Although unnecessary, in order to make the cycle stand out, dotted vertical lines are drawn between dots sharing a breakpoint in the sequence. (Note: from now on a *cycle of length n* will be called a *n-cycle*).

In this context we might realize that single dots can be seen as cycles of length 1 (after 4 → before 5, we just join a dot with itself).

This is very good news, we just made the absence of breakpoint the elementary variant of a bigger concept, it means we extracted information of the same kind that is also more subtle.

The completed graph for the sequence is shown in figure 1.3 (I recall that the dot before 1 is joined with one after the first ∘ symbol, and the dot after 10 is joined with one before the second ∘ symbol).

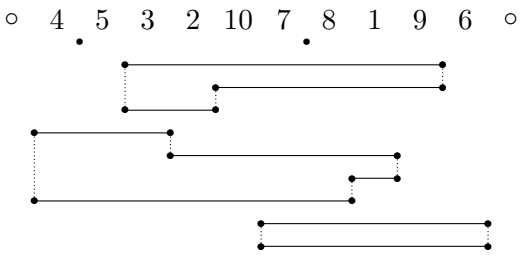


Figure 1.3: The completed graph contains two 1-cycles, one 3-cycle, one 4-cycle and one 2-cycle

1.4.3 Using the graph

Let's summarize once again:

- In the sorted sequence we should have eleven 1-cycles ($11 \times 1 = 11$).
- In the shuffled sequence we have two 1-cycles, one 3-cycle, one 5-cycle and one 2-cycle ($2 \times 1 + 3 + 4 + 2 = 11$).

The total amount of edges seems invariant and equal to the number of elements in the sequence + 1.

We know the starting graph, we know the goal graph. The problem just shifted from *sorting a sequence* to *transforming a graph*. In order to solve this, naturally we need to study how a BI will modify the graph at each step.

It suffices to draw the graphs for each successive state of the sequence. By doing so, one will notice that each BI extracted two 1-cycles from other cycles:

- the first BI extracted two 1-cycles, one from the 4-cycle and another one from the 2-cycle (thus leaving as remainder a 3-cycle and **a 1-cycle**).
- the second BI extracted two 1-cycles from the two 3-cycles (leaving two 2-cycles as remainder).
- the last BI extracted two 1-cycles from the two 2-cycles (**leaving two 1-cycles as remainder**)

In conclusion, sorting the sequence is the act of breaking down all cycles, extracting two 1-cycles with each BI.

Each bigger cycle will eventually give us an extra 1-cycle as remainder, which is like a half-bonus (a BI extracts two 1-cycles, so a 1-cycle remainder is like half a BI for free).

1-cycles already present in the graph also count as half-bonuses since they are operations we don't need to perform.

Therefore, there are as many half-bonuses as there are cycles, and we can derive an exact formula for the minimum number of BI required to sort any sequence G:

$$d_{BI}(G) = \frac{n+1-C}{2}$$

n is the number of elements in the sequence G (I recall we are working with $n+1$ edges in total), C is the number of cycles in the graph (and the formula is divided by 2 because a BI extracts two 1-cycles).

1.5 Genome rearrangements

If we label each gene or group of genes (we will use the term *marker*) with integers, then genomes can be seen as sequences of integers, and in this context, changes they undergo during evolution can be seen as operations on integer sequences.

Sorting a sequence into numerical order also allows us to study the distance between any two given genomes, as it's a matter of relabeling the elements in both genomes so that the second one is in numerical order, as illustrated in figure 1.4.

(∘ 2 8 5 3 9 7 6 4 10 1 ∘)	(∘ 4 5 3 2 10 7 8 1 9 6 ∘)
⇓ ?	⇓ ?
(∘ 4 3 5 2 8 1 7 6 10 9 ∘)	(∘ 1 2 3 4 5 6 7 8 9 10 ∘)

4→1; 3→2; 5→3; 2→4; 8→5; 1→6; 7→7; 6→8; 10→9; 9→10

Figure 1.4: Rewriting the labels

This is, of course, considering that each marker appears **exactly once** in both genomes. When markers appear multiple times (when there are *replicated markers*), rearrangement problems usually become harder. My work was to specifically design and study such harder rearrangement problems.

1.6 Closing words and a bit of philosophy

This intro is a good example of the general line of thought I used in my contributions.

Theoretical computer science (and more generally math) is an art of shapeshifting: it teaches us that everything is a model, nothing but representation of information, that could be rewritten in countless other forms. Solving an equation, or proving a theorem, can be seen as a matter of rewriting information into another form that will make the answer more explicit.

Because optimally sorting using a set of operations is always a process full of subtleties that need to be explicited, in genome rearrangement, we usually use data structures as a mean of rewriting information, as a tool of elegancy.

While it would have been possible to find a formula and proving it without the need for a graph structure, it wouldn't have been so simple, it wouldn't have been as useful in terms of comprehension (I think *sorting is the act of breaking down cycles* is a result that suddenly allows a complete understanding, getting rid of any complex implicit behavior), and of course the formula wouldn't have been so simple either.

The "hard" part I did not include here in order to let this section remain a *playful* introduction is mathematically proving that there always exist a BI that will successfully extract two 1-cycles from the graph (it could be proved with another data structure meant to simplify the proof, in similar fashion to the proof I give for "single tandem-halving by BI" later in the present document).

I'd like to give credit to Anne Bergeron as I am using her signature style of graph, drawing dots under the sequence rather than the traditional vertices and edges (I've always found it was the best way to keep the cycles apparent enough while displaying the sequence at the same time, so thanks for her idea).

Chapter 2

State of the art

The goal isn't to be exhaustive, but rather to provide a little bit of context to better understand my contribution, what was already done, what was being done and what wasn't done yet when I worked on genome rearrangements.

2.1 Notations (I) - Genome, markers, adjacencies, extremities, breakpoints

Rearrangement problems deal with successive transformation of genomes, at the scale of *genomic markers* (ie. genes or groups of genes).

It means in our models a genome consists of linear or circular chromosomes that are composed of genomic markers. Markers are represented by signed integers such that the sign indicates the orientations of markers in chromosomes. As there are only 2 possible orientations, naturally we have $--x = x$. A linear chromosome is represented by an ordered sequence of signed integers surrounded by the unsigned marker $\circ$ at each end indicating the telomeres (chromosome extremities). A circular chromosome is represented by a circularly ordered sequence of signed integers. For example, $(1\ \ 2\ \ -3)$ $(\circ\ \ 4\ \ -5\ \ \circ)$ is a genome composed of one circular and one linear chromosome.

An *adjacency* in a genome is a pair of consecutive markers. Since a genome can be read in two directions, the adjacencies $(x\ \ y)$ and $(-y\ \ -x)$ are equivalent. For example, the genome $(1\ \ 2\ -5)$ $(\circ\ \ -3\ \ 4\ \ 6\ \ \circ)$ has seven adjacencies, $(1\ \ 2)$, $(2\ -5)$, $(-5\ \ 1)$, $(\circ\ \ -3)$, $(-3\ \ 4)$, $(4\ \ 6)$, and $(6\ \ \circ)$. When an adjacency contains a $\circ$ marker, *i.e.* a telomere, it is called a *telomeric adjacency*.

When needed, we will refer to marker extremities directly, indicating them using a dot. Thus, adjacency $(x\ \ y)$ concerns extremities $x\cdot$ and $\cdot y$.

2.2 Distance and scenario

Usually there are two problems to solve in rearrangements. Finding the *minimal distance*, and finding a *minimal scenario*.

The minimal distance is the minimum number of operations required to go from the input

genome to a desired solution. In all models studied in this thesis this is a distance in the mathematical sense.

A minimal scenario is a sequence of operations transforming the input genome into a solution, whose length (number of operations) is the minimal distance.

While computing a scenario generally takes more time than computing the distance, in most operation models both problems belong to the same complexity class[2].

2.3 Simple markers

Even though I worked exclusively on genomes with duplicated content, I will start describing rearrangement problems where genomes have only one copy for each marker, as not only it provides context, but also helps understanding duplicated genomes problems better.

The problem is usually to find a way to transform a genome into the identity permutation (this allows to transform any genome into any other, as explained in chapter 1)

2.3.1 Breakpoint distance

Intro

The breakpoint distance is a measure based on the number of breakpoints between two genomes, or between a genome and the identity permutation, as it was done in the beginning of chapter 1. There are no operations associated with it and thus no scenario.

Example

Breakpoint distance	Generalized breakpoint distance
$(\circ\ 1\ \ 2\ \blacktriangle\ -4\ \blacktriangle\ -5\ \blacktriangle\ 3\ \blacktriangle\ \circ)$	$(\circ\ 1\ \ 2\ \blacktriangle\ -4\ \blacktriangle\ -5\ \blacktriangle\ -7\ \triangle\ \circ)$ $(\ \blacktriangle 6\ \blacktriangle\ 3\ \blacktriangle\ -8\ \blacktriangle\ 9\ \blacktriangle\ -10\ )$
$d = 4$	$d = 8 + 0.5 = 8.5$

Note: In generalized breakpoint distance, a telomeric breakpoint (indicated in blue) is worth 0.5. Also, in our example the second chromosome is circular (there are no $\circ$ markers), therefore the breakpoint before 6 is not telomeric, it is a breakpoint between markers -10 and 6.

History and references

The breakpoint distance was first introduced as a lowerbound, a 2-approximation for the reversal distance [Kececioglu and Sankoff, 1993] (meaning the breakpoint distance is never more than twice the reversal distance). It was computed in $O(n)$, leading to a 2-approximate reversal scenario in $O(n^2)$.

[2] see section 3.2.2 for a quick proof

Generalized breakpoint distance

In its first form, the breakpoint distance was defined on permutations, ie. on unilinear genomes.

Much later, multichromosomal variants were proposed, notably one in [Tannier et al., 2009] for which a lot of previously NP-hard problems became polynomial just by allowing circular chromosomes in genomes (and considering telomeric adjacencies/breakpoints are worth **half** their regular counterpart).

While several open questions remained in this paper, another researcher did an impressive extensive work answering them in [Kovac, 2011].

Outro

While not attached to a particular operation, the breakpoint distance is useful as a lowerbound for more complex models.

The generalized variant of the distance shows some interesting results: while some usually NP-hard problems becoming polynomial is not a surprise as the model is much less accurate, a well-known polynomial problem became NP-hard in some cases[3].

The breakpoint distance was also used in the context of comparing genomes with differing content albeit proven NP-hard [Blin et al., 2004].

Along with the generalized breakpoint model, another breakpoint distance variant, the Single-cut-or-join (SCJ) was introduced [Feijão and Meidanis, 2009] [Feijao and Meidanis, 2011] [Biller et al., 2013], and further studied in other sources [Bérard et al., 2012] [Miklos et al., 2013]. This variant is similar to the other generalization in that it simplifies some NP-hard problems, but also has the benefit of having an associated operation model, and thus rearrangement scenarios.

2.3.2 Sorting by reversals

Intro

The reversal (or *inversion*) mechanism has been observed by biologists studying drosophila genomes [Sturtevant and Dobzhansky, 1936] [Dobzhansky and Sturtevant, 1938].

A reversal acts on a segment of the genome and inverts both the order and the sign of markers within the segment.

Example

$$(\circ\ 1\ 2\ -5\ \underline{-7\ -6\ 3\ 4}\ 8\ 9\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ \underline{-5\ -4\ -3}\ 6\ 7\ 8\ 9\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \circ)$$

[3] see genome halving in [Kovac, 2011]

History and references

As aforementioned, study of the reversal mechanism predates bioinformatics, and it took half a century before it were reformulated in terms of computer science [Watterson et al., 1982] [Sankoff, 1989].

At first though, only approximation algorithms were given and it wasn't even known whether the problem was polynomial. The answer came later with a $O(n^4)$ algorithm presented in [Hannenhalli and Pevzner, 1995a], revisited and simplified [Kaplan et al., 1997] [Bergeron, 2001] [Tannier and Sagot, 2004] down to a subquadratic $O(n^{\frac{3}{2}}\sqrt{\log n})$ algorithm [Tannier et al., 2007] making good use of an earlier data structure first introduced in [Kaplan and Verbin, 2005].

About the distance itself, without the need for a rearrangement scenario, an efficient linear-time algorithm was published [Bader et al., 2001] as well as another very elegant one, still in $O(n)$ [Bergeron et al., 2004].

The unsigned case

It was previously said that a reversal alters the order as well as the sign of markers. In the early days of sequencing, though, the sign couldn't be determined and thus the first algorithms considered unsigned permutations, meaning the sign was disregarded (another way to see it is that everything has a positive sign, and a reversal changes the order of markers but leaves the sign unchanged).

The problem of sorting unsigned permutations by reversals is a classical NP-hardness result in bioinformatics [Caprara, 1997].

It might sound like a counter-intuitive result if we consider the sign is an added constraint on the final configuration, but it is in fact more accurate to see it that way: it is a strong enough constraint to leave us no choice.

The sign gives us a valuable piece of information about the final position of a marker. Since our problems are about finding a *minimal* distance, disregarding the sign is equivalent to having to find the sign affectation that will minimize the distance.

Outro

Several generalizations and contraints have been proposed on this model such as perfect scenarios in polynomial time [Sagot and Tannier, 2005] [Bérard et al., 2007] [Bérard et al., 2008] (scenarios that respect conservation criteria making them more likely from a biological point of view), or other studies dedicated to the solution space [Braga et al., 2007] [Braga, 2009] just to name a few.

Another interesting extension was the *HP distance*, using reversals to simulate different operations on a multichromosomal genome, namely reversals and translocations. It was introduced in [Hannenhalli and Pevzner, 1995c] with a polynomial algorithm and a rather complex distance formula, further simplified [Ozery-Flato and Shamir, 2003] [Jean and Nikolski, 2007], and finally elegantly tackled in [Bergeron et al., 2008] by making good use of a bigger abstraction framework, the DCJ operation, introduced in [Yancopoulos et al., 2005].

2.3.3 Other operation models

Reciprocal translocation

We've briefly talked about the HP distance, using reversals on multichromosomal genomes to simulate other biological operations [Hannenhalli and Pevzner, 1995c]. One of the HP distance operations is the *reciprocal translocation* and it was also later studied as a standalone operation.

It consists in swapping telomeric extremities between chromosomes while reversing them.

$$(\circ\ 1\ \ 2\ \ 3\ \underline{-8\ -7\ -6\ \circ)\ \ (\circ\ -5\ -4}\ \ 9\ \ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \circ)\ (\circ\ 6\ \ 7\ \ 8\ \ 9\ \ 10\ \circ)$$

The model was introduced in [Kececioglu and Ravi, 1995] and a polynomial-time algorithm followed [Hannenhalli, 1995]. An error in that algorithm was corrected and a $O(n^3)$ algorithm was given [Bergeron et al., 2005].

Later, in [Ozery-Flato and Shamir, 2006], the efficient data structures developed for sorting by reversals were reused, leading to the same complexity for sorting by reciprocal translocations. The authors raised a mathematically interesting question: is it possible to linearly reduce one problem into the other?

To my knowledge this question remains open.

Block interchange

The block interchange operation has been used as an illustration in chapter 1. It is a swap of two blocks in the genome.

$$(\circ\ \boxed{6\ 7\ 8\ 9\ 10}\ 4\ 5\ \boxed{1\ 2\ 3}\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \circ)$$

It was introduced and first solved in [Christie, 1996]. Enhancements were later made in [Lin et al., 2005] and [Feng and Zhu, 2007] bringing the original $O(n^2)$ complexity down to a $O(n \log n)$ algorithm by using a permutation tree. The distance is linear.

Transposition

A transposition is an operation where a single block from the genome is moved somewhere else.

$$(\circ\ 1\ 2\ \blacktriangle\ 5\ 6\ 7\ 8\ 9\ \boxed{3\ 4}\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \circ)$$

It can also be seen as a restriction on block interchange: the two blocks have to be adjacent.

$$(\circ\ 1\ 2\ \boxed{5\ 6\ 7\ 8\ 9}\ \boxed{3\ 4}\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \circ)$$

First introduced in [Bafna and Pevzner, 1995], sorting by transpositions remained an open problem for 15 years, during which publications emerged to give us approximation algorithms [Bafna and Pevzner, 1998] [Hartman and Shamir, 2004].

Very recently an answer has been given in [Bulteau et al., 2010], in the form of a NP-hardness proof.

It is interesting to see that such a small constraint on block interchange makes all the difference (and that there are indeed NP-hard rearrangement problems with simple markers).

Prefix reversal

This problem is also known as *pancake flipping* in its original formulation [Dweighter, 1975].

Like with reversals, I will briefly talk both of the signed and unsigned cases.

A prefix reversal, as implied by its name, is a restriction on the reversal operation where the segment has to start at the beginning of the sequence.

In the unsigned case, once again, the sign is ignored.

$$\underline{(\circ\ 3\ 4\ 5}\ 2\ 1\ 6\ 7\ 8\ 9\ 10\ \circ)$$
$$\Downarrow$$
$$\underline{(\circ\ 5\ 4\ 3\ 2\ 1}\ 6\ 7\ 8\ 9\ 10\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \circ)$$

In the *unsigned* case, the problem is NP-hard [Bulteau et al., 2012] (let us remind that it was already the case with regular reversals).

Trivia surrounding this problem aludes to the fact the famous founder of Microsoft, Bill Gates, co-signed an academic paper devoted to this problem [Gates and Papadimitriou, 1979]. This is his only paper. In this article the authors also introduce the *burnt pancakes* variant which is equivalent to the signed case.

To this date the *signed* case remains an open problem. Polynomial subclasses of the problem have been shown as well as some bounds for general instances in [Labarre and Cibulka, 2011].

I have also briefly worked with Laurent Bulteau on a *prefix DCJ* model that surprisingly allowed us to reestablish the best known bounds on the prefix reversal model in a much simpler way. Unfortunately that quick overview did not allow us to go further than this point.

2.3.4 DCJ

Intro

A *DCJ* (double-cut and join) operation on a genome G cuts two different adjacencies in G and glues pairs of the four exposed extremities in any possible way, forming two new adjacencies. For

example, the following DCJ cuts adjacencies $(1\ \ 2)$ and $(-5\ \ 6)$ to produce $(1\ \ 6)$ and $(-5\ \ 2)$.

Example

Extremities are bi-colored to indicate how they are glued afterwards (joining the same color).

$$(\circ\ 1\ \ 2\ \ 8\ \ 9\ \blacktriangle\ -6\ \ -5\ \ -4\ \ -3\ \ -7\ \blacktriangle\ 10\ \circ)$$
$$\Downarrow$$
$$(-6\ \ -5\ \ -4\ \ -3\ \blacktriangle\ \ -7)\ (\circ\ \ 1\ \ 2\ \blacktriangle\ 8\ \ 9\ \ 10\ \ \circ)$$
$$\Downarrow$$
$$(\circ\ \ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ \ \circ)$$

History and references

The DCJ is more abstract than the other operations considered so far, and it allows more diversity in genome manipulation. Indeed, it is a genius generalization of all previous operations, first introduced in [Yancopoulos et al., 2005] and elegantly tackled in [Bergeron et al., 2006] giving us $O(n)$ algorithms for both distance and scenario in the signed case.

The unsigned case has been proven NP-hard [Chen, 2010].

Outro

As seen in the illustration, some DCJ operations can extract content into a new chromosome (*excisions*), or merge chromosomes together (*reintegrations*).

Because manipulating the chromosomal topology too much isn't very reasonable from a biological point of view, a restricted model was introduced at the same time [Yancopoulos et al., 2005], where a chromosome excision would have to be followed by its immediate reintegration. The model has later been further studied and better algorithms were proposed [Kovác et al., 2010] [Kováč et al., 2011].

Another example of extension of the DCJ model is sorting between genomes with differing contents [Yancopoulos and Friedberg, 2008] [Braga et al., 2010], even in the restricted model [Braga and Stoye, 2013].

Just like it was the case for the reversal model, the solution space of DCJ sorting has also been studied by the same author [Braga and Stoye, 2010][4].

The operation model itself was also further generalized in [Alekseyev and Pevzner, 2008], where the authors expand on the DCJ model, seeing it as the particular case for what they call a *k-break* operation (which cuts the genome at k positions then glues all exposed extremities in any possible way) where $k = 2$.

2.3.5 Phylogeny

Originally, rearrangement problems were introduced as the elementary step of a more ambitious project: reconstructing phylogeny.

[4] ...and "independently" by one of my former advisors, although in a very rushed and inferior way...

The principle is to use evolutionary distances as a way to measure relative distance between genomes, and rebuilding the phylogenetic tree (all common encestors and their relative positioning) from nothing but the set of present genomes.

The *large phylogeny* problem aims at reconstructing the whole phylogenetic tree topology as well as all common ancestors from a given set of genomes.

2.3.5.1 Small phylogeny and median

Intro

Small phylogeny is an "easier" variant of the problem, where the tree topology is also given as an input [Sankoff and Blanchette, 1998].

Given a tree topology as well as the genomes present at its leaves, ancestor genomes should be inferred using a parsimony criterion: the sum of distances on all branches of the tree has to be minimal.

The *median* problem is a further simplified variant of small phylogeny, the special case where there are only 3 genomes and one common ancestor to infer to them all.

NP-hardness of the median problem would naturally imply small phylogeny and large phylogeny NP-hardness as well.

History and references

The median problem has been proven NP-hard under most rearrangement models (breakpoint [Pe'er and Shamir, 1998, Bryant, 1998, Tannier et al., 2009], reversals [Caprara, 2003], and DCJ [Tannier et al., 2009]), although in practice we have good branch and bound algorithms for computing optimal medians.

Under the generalized breakpoint model, when circular chromosomes are allowed, the problem becomes polynomial as demonstrated by a $O(n^3)$ algorithm [Tannier et al., 2009], then a better $O(n \log n)$ algorithm [Kovac, 2011].

Even though this was a very promising result, small phylogeny remains NP-hard under this model, even for as little as 4 species.

Outro

The median problem was also studied under SCJ, the other generalized breakpoint distance.

Under this model the median is computable in $O(n)$, even in the multilinear case, and small phylogeny is polynomial too [Feijão and Meidanis, 2009, Feijao and Meidanis, 2011]. This result has been experimentally used with promising results [Biller et al., 2013]. Large phylogeny remains NP-hard, though.

Another very constrained reversal model (constrained in terms of allowed chromosomal topology as well as constrained specific reversals) allowed a polynomial answer, namely a $O(n)$ reversal median problem [Ohlebusch et al., 2005].

2.4 Notations (II) - Duplicated genomes, double-adjacencies

A duplicated genome contains at most two occurences of each marker. Two copies of a same marker in a genome are called paralogs. If a marker x is present twice, one of the paralogs is represented by $\overline{x}$. By convention, $\overline{\overline{x}} = x$.

Definition 1 *A* duplicated genome *is a genome in which a subset of the markers are duplicated.*

For example, $(1\ \ 2\ \ -3\ \ -\overline{2})\ (\circ\ \ 4\ \ -5\ \ \overline{1}\ \ \overline{5}\ \ \circ)$ is a duplicated genome where markers 1, 2, and 5 are duplicated. A *non-duplicated genome* is a genome in which no marker is duplicated. A *totally duplicated genome* is a duplicated genome in which all markers are duplicated. For example, $(1\ \ 2\ \ -\overline{2})\ (\circ\ \ -3\ \ \overline{1}\ \ \overline{3}\ \ \circ)$ is a totally duplicated genome.

A *double-adjacency* in a genome G is an adjacency $(a\ b)$ such that $(\overline{a}\ \overline{b})$ or $(-\overline{b}\ -\overline{a})$ is an adjacency of G as well. Note that a genome always has an even number of double-adjacencies. For example, the four double-adjacencies in the following genome are indicated by $\diamond$:

$$G = (\circ\ \ 1\ \ \overline{1}\ \ 3\ \ 2\ \diamond\ 4\ \diamond\ 5\ \ 6\ \ \overline{6}\ \ 7\ \ \overline{3}\ \ 8\ \ \overline{2}\ \diamond\ \overline{4}\ \diamond\ \overline{5}\ \ 9\ \ \overline{8}\ \ \overline{7}\ \ \overline{9}\ \ \circ)$$

Definition 2 *A* perfectly duplicated genome *is a totally duplicated genome such that all adjacencies are double-adjacencies, none of them in the form* $(x\ -\overline{x})$.

For example, the genome $(1\ \ 2\ \ 3\ \ 4\ \ \overline{1}\ \ \overline{2}\ \ \overline{3}\ \ \overline{4})$ is a perfectly duplicated genome, while $(\circ\ \ 1\ \ 2\ \ -\overline{2}\ \ -1\ \ \circ)$ is not. (Note: this definition is equivalent to the one from [Mixtacki, 2008]).

2.5 Duplicated content

As aforementioned, all previously cited work concern a sub-class of genomes: genomes for which each marker appears only once.

This is in fact far from biological reality where genes can appear in multiple copies, and naturally several problems taking this into account have been designed and solved, while simple markers problem could be seen as a first step. As we are about to see, the next step is far from being trivial, from a computational point of view, even under the strong assumption that each marker can only appear twice in a genome.

2.5.1 Exemplar distance and matching models

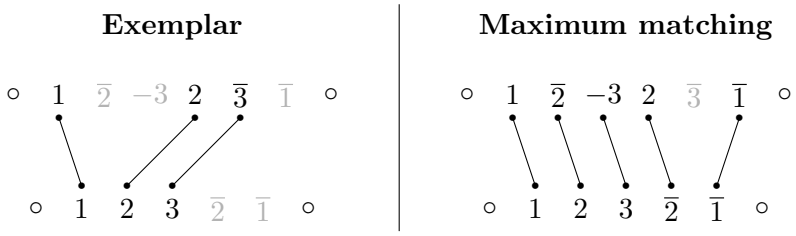
Intro

As computer scientists, a natural way to tackle duplications is to find a reduction to a simple marker problem. If one were able to distinguish copies of a marker, then they could be labeled as 2 distinct ones and the problem remains the same as its simple markers variant.

Seeking to keep the maximum number of genes in both genomes while giving them a one-to-one correspondance is known as the *maximum matching model* [Blin et al., 2004].

Another approach is to keep just one copy of each marker and see which choices allow the minimal distance: this is the *exemplar model* [Sankoff, 1999].

Example



History and references

The *exemplar model* was first introduced under the breakpoint and the signed reversal distances [Sankoff, 1999], along with branch-and-bound algorithms, while the exact complexity was not settled.

A NP-completeness proof for both distances was given in [Bryant, 2000].

The *maximum matching* model was introduced in [Blin et al., 2004] as a mean to study rearrangements where reversals, insertions and deletions are allowed, but it was proven NP-hard. Computing breakpoint distance or reversal distance under maximum matching is also NP-hard [Chen et al., 2005].

In [Blin et al., 2007], other comparison measures have also been proven to be NP-hard for both models.

A third matching model, the *intermediate matching* was introduced as a bridge between both models [Angibaud et al., 2007], where *at least one* copy of each marker is kept.

Naturally, since both the exemplar and the maximum models are particular cases of the intermediate model, NP-hardness remains.

Outro

Solved problems for simple markers become NP-hard as soon as duplications are introduced, since otherwise it would imply a polynomial-time matching.

Further restrictions of the exemplar distance have been studied, such as the *zero exemplar distance problem*: Is the distance equal to zero? i.e. can two genomes be reduced to the same one via an exemplar matching?

Even this strong restriction is NP-hard for two monochromosomal genomes with as little as at most two occurences of each marker in both of them [Blin et al., 2009], or even disregarding gene sequences and only considering chromosomes as unordered sets of markers [Jiang, 2011]. It becomes polynomial only for a stronger restriction: each gene has to appear exactly once in one of the two genomes, and at least once in the other [Jiang, 2011].

2.5.2 Genome Halving

Intro

In an attempt to avoid NP-hardness, the Genome Halving problem does not constrain the final configuration.

Given a duplicated genome G, we assume that all duplications are the result of a single *whole genome duplication* (WGD) event. The goal is to find the non-duplicated ancestor genome, ie. the state in which the genome was just before the WGD occured.

Naturally, it is done under the hypothesis of parsimony, meaning we want to minimize the distance between the genome and its ancestor.

Example

$$G = (\circ\ 1\ \ 2\ \ -\overline{3}\ \circ)\ (\circ\ \overline{1}\ \ 3\ \blacktriangle\ 4\ \ \overline{5}\ \blacktriangle\ -\overline{2}\ \circ)\ (\ \overline{4}\ 5\ )$$
$$\Downarrow$$
$$(\circ\ 1\ \blacktriangle\ 2\ \ -\overline{3}\ \blacktriangle\ \circ)\ (\ 4\ \ \overline{5}\ )\ (\circ\ \overline{1}\ \ 3\ \ -\overline{2}\ \circ)\ (\ \overline{4}\ 5\ )$$
$$\Downarrow$$
$$G' = (\circ\ 1\ \ \overline{3}\ \ -2\ \circ)\ (\ 4\ \ \overline{5}\ )\ (\circ\ \overline{1}\ \ 3\ \ -\overline{2}\ \circ)\ (\ \overline{4}\ 5\ )$$

Note that the biological scenario really is *backwards*: chronologically, there was a non-duplicated genome $G'' = (\ 1\ \ \overline{3}\ \ -2\ )\ (\ 4\ \ \overline{5}\ )$ which underwent a WGD and thus became the perfectly duplicated genome G', then other rearrangements happened so that we finally observe G.

History and references

Genome Halving has first been studied under reversals [El-Mabrouk et al., 1998] and HP distance [El-Mabrouk and Sankoff, 2003, Alekseyev and Pevzner, 2007], all with polynomial algorithms as answer.

The DCJ variant was studied in [Warren and Sankoff, 2008], and brilliantly in [Mixtacki, 2008] yielding a linear-time algorithm.

Surprising results arise with the generalized breakpoint distance model: while it remains polynomial for multichromosomal genomes with circular chromosomes allowed [Tannier et al., 2008], it is NP-hard for monochromosomal and multilinear genomes [Kovac, 2011].

Outro

The genome halving problem was a breakthrough, being the first polynomial duplicated rearrangement problem.

However, its polynomiality can be accounted for by the fact there is an exponential amount of optimal solutions, coupled with an exponential amount of possible scenarios to each of them.

Loosely speaking, the genome halving problem allows to sort half of the adjacencies so that they mimick the adjacencies formed by their paralogs. It is conceptually very close to a simple marker rearrangement problem.

Needless to say, so many different genomes being optimal also raises realism issues from a biological point of view.

2.5.3 Other classical problems

Intro

The genome halving problem led to two other classic problems:

The *genome aliquoting* is a generalization of genome halving, with more than two copies per marker.

The *guided halving* is a problem designed to answer the realism issues of genome halving. By providing a *reference* genome in addition to a duplicated genome, we now look for a perfectly duplicated genome that minimizes the sum of its distances towards the duplicated genome and the reference genome.

Genome aliquoting

The genome aliquoting problem was first introduced in [Warren and Sankoff, 2009] along with a heuristic algorithm as a first attempt to provide a solution. Further studies were made by the same authors using the generalized breakpoint distance as a 2-approximation for the DCJ distance [Warren and Sankoff, 2011].

The problem remains open as of today.

Guided halving

Using an external reference genome to constrain solutions to the genome halving was first introduced in [Zheng et al., 2006] along with a heuristic algorithm, then proven NP-hard, even under breakpoint distance [Zheng et al., 2008].

Under generalized breakpoint with circular chromosomes allowed, however, it becomes polynomial, first solved in $O(n^3)$ [Tannier et al., 2009], further enhanced to $O(n \log n)$ [Kovac, 2011].

On the whole this problem seems to display the same complexity as the median problem, which is not so surprising given conceptual similarities between the two problems[5].

Outro

While the genome halving was a promising start, it was drifting away from biological reality, and both attempts at solving those issues are NP-hard.

Was the genome halving a breakthrough, or was it a deadend? Was its polynomiality the fleeting ray of light reinforcing the notion of darkness?

2.6 Closing words

In conclusion, rearrangement problems are defined on 3 main parameters:

- type of input genome(s)
- allowed operations
- desired configuration

A recent analysis of rearrangement with duplications has been published by experts on the field [El-Mabrouk and Sankoff, 2012], which is a much recommended reading for anyone interested in the field.

[5] NP-hardness was proven by reduction from the median problem

Chapter 3

Rearrangements with duplicated markers

In this section I study rearrangement problems in the vein of genome halving.

The difference between genome halving and the problems I studied is that, rather than basing the problem on "whole genome duplication" events, I studied segmental duplications instead, under various assumptions. I also studied another model where duplications occur on-the-fly as rearrangement operations happen.

For segmental duplications, I proved that a single tandem could be reconstructed in polynomial time. I provided $O(n^2)$ algorithms for the scenario, and an $O(n)$ computation for the distance, for both the DCJ and Block Interchange operation models.

I also proved NP-hardness of several problems based on multiple tandem reconstruction.

The breakpoint duplication model is also proved NP-hard, and I designed a FPT algorithm in the number of cycles present in a graph made for it.

These analyses led to several publications, in [Thomas et al., 2011] [Thomas et al., 2012a] [Thomas et al., 2012b] and [Thomas et al., 2013].

3.1 Preliminary game II: genome halving

It would benefit the reader and generally help the self-containment of this thesis to briefly revisit the classical genome halving by DCJ problem as it was handled in [Mixtacki, 2008], because I use this problem as a starting point for the single tandem halving problem developed in section 3.4, and because it provides good illustrations to what I am about to develop in section 3.2.1.

Similarly to what was done earlier in chapter 1, I will focus on general workings rather than technical details. Since the notations have already been introduced previously, we shall dive even faster into the subject, considering the reader to already be familiar with genome rearrangements at this point.

Although what follows is similar to the analysis framework of chapter 1, it is also slightly more complex as the genome halving problem presents some additional subtleties.

3.1.1 Rules and example

I recall the genome halving by DCJ problem, using terms defined in section 2.4.

The input genome is a *totally duplicated genome* (each marker appears exactly twice), and the goal is to find a closest (with respect to the DCJ distance) perfectly duplicated genome (each adjacency must be a double-adjacency, none of them them in the form $(x\ -\overline{x})$).

Here is an example halving scenario.

$$(\circ\ 1\ \blacktriangle\ \overline{4}\ \ \overline{1}\ -2\ \overline{2}\ \blacktriangle\ 3\ \ -4\ \ \overline{3}\ \circ)$$
$$\Downarrow$$
$$(\circ\ 1\ -\overline{2}\ 2\ -\overline{1}\ \blacktriangle\ -\overline{4}\ 3\ -4\ \overline{3}\ \blacktriangle\ \circ)$$
$$\Downarrow$$
$$(\circ\ \blacktriangle\ \circ)\ (\circ\ 1\ \ -\overline{2}\ \blacktriangle\ 2\ -\overline{1}\ \circ)\ (\ -\overline{4}\ \ 3\ \ -4\ \ \overline{3})$$
$$\Downarrow$$
$$(\circ\ 1\ \ -\overline{2}\ \circ)\ (\circ\ 2\ -\overline{1}\ \circ)\ (\ -\overline{4}\ \ 3\ \ -4\ \ \overline{3})$$

Note that markers don't necessarily have to end in consecutive order, as it might not always be the shortest way to go.

Also, about the last performed operation in this example, it is a *fission*. With our usual DCJ definition, we have to consider that the second breakpoint is in fact in an empty linear chromosome (which is a theoretical tool rather than an actual part of the genome). You can ignore it for now as it is much easier to explain in a graph, as we are about to see.

3.1.2 General case

3.1.2.1 Drawing the *natural graph*

Because we are aiming at a closest genome satisfying a *pattern* (namely, *any* perfectly duplicated genome), the exact goal genome is not known, and thus we cannot build a graph relying on "what we *should* obtain" as it was done in chapter 1.

However, we will still go in a generally similar line of reasoning as we are about to design a graph such that any perfectly duplicated genome is always corresponding to a particular configuration in terms of connected components.

By definition, a perfectly duplicated genome is a genome such that every adjacency is a double-adjacency. It follows that if we use edges to join paralogous extremities, a double adjacency can easily be identified, as shown in figure 3.1. The graph defined this way is called the *natural graph*.

In the natural graph, a telomeric double-adjacency is necessarily a 1-path, while double-adjacencies concerning two distinct markers are 2-cycles.

Thus, any natural graph comprised of only 1-paths and 2-cycles is the natural graph of a perfectly duplicated genome.

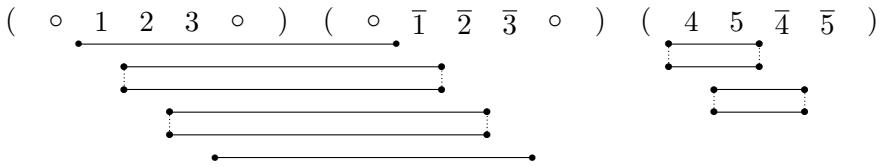
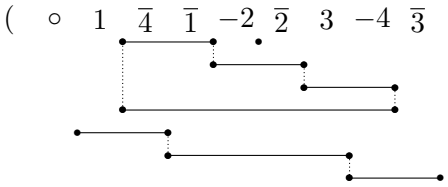


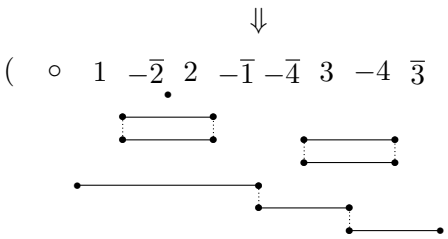
Figure 3.1: Natural graph of a perfectly duplicated genome. It consists of 1-paths and 2-cycles only.

3.1.2.2 Using the graph I: DCJ on a graph

Here are the successive natural graphs for the above scenario example. It will prove useful for studying how a DCJ can alter the graph.

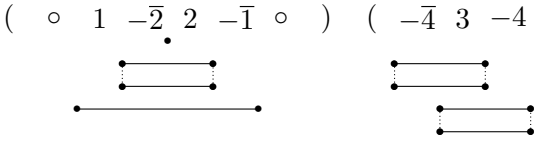


Contents: one 3-path, one 1-cycle and one 4-cycle.



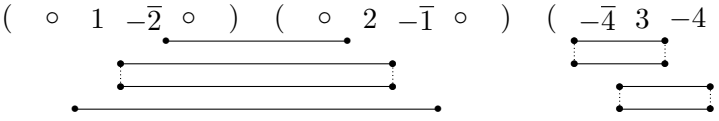
A DCJ extracted a 2-cycle from the 4-cycle (remainder: **2-cycle**)

Contents: one 3-path, one 1-cycle and two 2-cycles.



A DCJ extracted a 2-cycle from the 3-path (remainder: **1-path**)

Contents: one 1-path, one 1-cycle, three 2-cycles.



A DCJ transformed the 1-cycle into a 1-path (remainder: none)

Contents: two 1-paths, three 2-cycles.

As the nodes from the graph are corresponding to adjacencies of the genome, a DCJ operation, from the graph point-of-view, will cut connected components at 2 positions and glue the exposed extremities in any possible way.

If you have trouble visualizing it, imagine the edges are pieces of string maintained together by blu-tack, the nodes. A DCJ essentially rips two blu-tack pieces in half before joining the four resulting ripped pieces in another way, changing the way the pieces of string are connected together.

Back to our graph, it means that a DCJ can extract a cycle from any bigger component, or merge two components together, depending on whether the two breakpoints are shared by the same component or not.

Note that you also have the right to consider the two positions are degenerated (two breakpoints on the same node), as illustrated in the last step of our example. This allows a DCJ to transform a path into a cycle or vice versa, or to split a path into two smaller paths.

3.1.2.3 Using the graph II: halving distance

The analysis will be done in two steps, in the same vein as chapter 1. First we will find what is invariant as it will serve as a bound for the formula, then we will inspect more closely how this bound relates to the exact distance by reasoning on the kind of connected components we are trying to reconstruct.

Invariant

The number of edges in the graph is invariant and equal to the total number of markers. We will write it as $2n$, with n the number of *distinct* markers.

We aim at reconstructing double-adjacencies, and a double-adjacency is a 2-cycle in the graph. Therefore, when bigger components are present in the graph, extracting 2-cycles from them are good operations. Since extracting a 2-cycle takes care of two edges from the graph, we can expect at most $\frac{2n}{2} = n$ operations, which gives us a first upperbound on the distance, $d \leq n$.

Just like it was already the case in chapter 1, computing the exact distance will be done by reasoning on the **remainders** left by our operations.

2-cycles

A DCJ can always extract a 2-cycle from any bigger component, and bigger *even* cycles will eventually give an additional 2-cycle as remainder (see step 1 of our example). That is like one operation for free, or in other terms, a bonus of one operation. 2-cycles already present in the graph count as bonuses as well, since they are also operations we don't need to perform.

$\Rightarrow$ If the natural graph contains EC even cycles, then there are EC 2-cycles which either are already present, or will eventually be formed as remainders of other 2-cycle extractions. Thus a better upperbound on the distance is $d \leq n - EC$

1-paths

I said that double-adjacencies were 2-cycles in the graph. While this is true for adjacencies concerning two distinct markers, in the case of telomeric adjacencies, they are 1-paths. This means that 1-paths already present and 1-paths obtained as remainders also contribute to the distance formula.

A single DCJ could create two 1-paths at once (by splitting a 2-path). It follows that 1-paths already present only count as half a bonus each[6]. Bigger odd-paths would also each eventually give a free 1-path as remainder.

$\Rightarrow$ If there are OP odd paths in the graph, then there are OP 1-paths that are either already present or will be formed as remainders of 2-cycle extractions. Since a 1-path is a bonus of half an operation, that makes $\frac{OP}{2}$ operations for free. It follows an even better upperbound would be $d \leq n - EC - \frac{OP}{2}$.

Distance

There is one last subtlety we need to address in order to express the distance.

In the case where there is an odd number of odd paths, only an even number of them will effectively give bonuses, because the remaining one eventually induces a remaining 1-cycle somewhere else (indeed, the total number of edges is even), which forces us to perform an operation to transform it into a 1-path (see step 3 of our example). It is to note that such operation affects only one edge from the graph, instead of the usual two. This is basically equivalent to a penalty of half an operation, which cancels the bonus we got through the 1-path remainder. Therefore the exact contribution of odd paths is rather $\lfloor \frac{OP}{2} \rfloor$.

Note that odd cycles and even paths cannot give us bonuses since perfectly duplicated genomes graphs cannot contain any. Thus, there exists no other way to save operations, and we indeed have the exact distance.

$\Rightarrow$ In conclusion, the distance formula is equal to $d = n - EC - \lfloor \frac{OP}{2} \rfloor$.

3.2 Meta-problems: general results

3.2.1 Dual layered vision of rearrangement problems

In this section I describe a vision of rearrangement problems which helped me obtaining results and that I have not seen described elsewhere. This is essentially a generalization of the analysis framework I used in my revisits of [Christie, 1996] and [Mixtacki, 2008].

I will first recall how the breakpoint distance can be used to establish bounds, as it is a prerequisite.

3.2.1.1 Bounds

Following the same logic used in chapter 1, if an operation affects two breakpoints, then a single operation can reconstruct a *maximum* of two adjacencies. This observation alone provides a

[6] note that this is also consistent with the generalized breakpoint distance where telomeric adjacencies count for 0.5

lowerbound on the distance.

Conversely, by looking at the *minimum* number of adjacencies that can always be reconstructed by an operation, an upperbound can be established. This upperbound is also useful to define approximation ratios.

For example, because a DCJ cuts the genome at two positions and is able to glue the exposed extremities in any desired way, it can always reconstruct one adjacency, and sometimes two, which implies that the breakpoint distance (number of adjacencies to be reconstructed) serves not only as an upperbound to the distance, but also as a 2-approximation[7] since in the best case two adjacencies can be reconstructed at each step yielding a distance that is half the breakpoint distance.

The same reasoning also holds for more general operation models as long as they are defined on a fixed number of breakpoints, like the multi-break model from [Alekseyev and Pevzner, 2008] (I recall that a k-break operation cuts the genome at k positions and glue all exposed extremities in any possible way, which means a DCJ can be seen as a 2-break operation).

Indeed, the breakpoint distance serves as a k-approximation for k-break operations (in the best case, each k-break operation succesfully reconstructs k adjacencies, leading to a distance k times smaller than the breakpoint distance).

3.2.1.2 From bounds to distance

We have seen that operations can be defined on a number of breakpoints, however it happens that different operations are defined on a same number of breakpoints yet act differently on them. To take this into account, one might say an operation is described on two levels:

- the lower level, related to the number of breakpoint it affects.
- the higher level, related to the way these breakpoints are affected.

For example, a reversal is defined on *two* breakpoints on a same linear or circular chromosome. The way these breakpoints are affected is by reversing the segment contained within, leaving the chromosomal structure intact. A DCJ, however, while also defined on two breakpoints, doesn't constrain the breakpoint selection (they don't need to be on a same chromosome), and furthermore any possible recombination of exposed extremities is allowed, which can lead to a change in chromosomal structure (a linear chromosome can be transformed into a linear and a circular chromosome, for example).

Based on this description, bounds are related to this lower level, while data structures are used to better study the higher level, allowing us to uncover what separates the distance from its bounds.

In this context, a distance formula can be seen as

$$d = (\text{number of expected operations}) - (\text{sum of bonuses})$$

The *number of expected operations* is a bound. It is the lower level contribution to the distance. It answers the question *"how many operations are needed in the worst case, given how*

[7] I recall the breakpoint distance was first introduced as a 2-approximation for the reversal distance in [Kececioglu and Sankoff, 1993]

many adjacencies there are to be reconstructed and how many of them can always be reconstructed by an operation?".

Note that this number is an invariant. It only depends on the problem, not on the considered genome, and thus it remains constant during a rearrangement scenario.

For example, in chapter 1, BI were the only allowed operations. For a genome of length n, there were $n + 1$ adjacencies to be reconstructed in the worst case (telomeric adjacencies were not given a different weight in this model since the problem was defined on unilinear genomes only). A single BI can always reconstruct at least 2 adjacencies. It follows that the number of expected operations in this case is $\frac{n+1}{2}$.

With my other example, on duplicated markers, in section 3.1, for a genome consisting of n markers each appearing twice, there are at most n adjacencies to be reconstructed (even though the genome length amounts to a total of $2n$ adjacencies, half of them can be left untouched, using the paralogs to copy them). A DCJ, which is the only allowed operation here, can always reconstruct one adjacency. It follows the number of expected operations in this case is n.

The sum of bonuses is what separate the bound from the actual distance. It depends on the genome, and it is the number of times we will be able to reconstruct more adjacencies in a fortuitous way during a scenario. It is usually more apparent on the graph (through *remainders* of operations) as it depends on the exact transformation applied, and this is why it is the higher level contribution to the distance.

I will keep the same two examples to illustrate this part of the formula:

In chapter 1, each sorting operation extracts two 1-cycles from bigger cycles. It follows that 1-cycles already present in the graph could be seen as the result of previously performed operations, and since an operation would create two of them, observing one is like observing the result of half an operation. In other words, we might say each 1-cycle already present is a bonus of half an operation. Bigger cycles will also each eventually give a 1-cycle as remainder, which is half a bonus too.

It follows that each cycle is half a bonus, so the *sum of bonuses* is $\frac{C}{2}$, with C the number of cycles in the graph.

In conclusion, as the number of expected operations was $\frac{n+1}{2}$, the distance formula becomes $\frac{n+1-C}{2}$.

In section 3.1, the goal graph is consisting of 2-cycles and 1-paths. A DCJ would extract a 2-cycle, therefore 2-cycles already present are a bonus of one operation each. Bigger *even* cycles eventually give a 2-cycle as remainder. It follows the number of even cycles, EC, contributes to the sum of bonuses.

With 1-paths it's a bit more subtle as previously explained. Since a single DCJ could create two 1-paths at once (by splitting a 2-path), they can only count as half a bonus each. Through extraction of 2-cycles, bigger *odd* paths will also each give a 1-path as remainder, eventually. It is also important to note that in the case where there is an odd number of odd paths, one of the paths would not contribute (refer to section 3.1 for details). It follows that, with OP being the number of odd paths, $\lfloor \frac{OP}{2} \rfloor$ also contributes to the sum of bonuses for genome halving.

In conclusion, as the number of expected operations was n, the distance formula becomes $n - (EC + \lfloor \frac{OP}{2} \rfloor)$.

In addition to helping to establish distance formulas, such vision also helps grasping a better understanding of scenarios construction, which might ultimately prove useful when studying

solution spaces, for example. Indeed, any sorting operation must necessarily increase the sum of bonuses.

3.2.1.3 On complexity

The lower level contribution is directly related to the breakpoint distance, which is usually polynomial even for problems that become NP-hard for more elaborate operation models.

It follows that the NP-hard part of such problems lies in computing the sum of bonuses.

Separating the NP-hard part from the polynomial part of a rearrangement problem allows for an easier reduction, and might even lead to FPT algorithms if the sum of bonuses is independent from the genome length.

For example, later in the present document, namely section 3.3.3, I study and prove NP-hardness of a rearrangement problem which would be a very good illustration of this principle. Obviously I cannot expand too much on it for now since the problem hasn't been defined yet, but I'll just say that the distance formula has the usual n as number of expected operations, while the number of bonuses, C_i is the direct result of a well-known NP-hard problem, allowing for a straightforward reduction. I will get back to it when the problem is defined: in section 3.3.4.3 I will show how to use the ideas from the present section to quickly establish the distance formula.

In conclusion, thinking in terms of "what is the number of expected operations" and "what are the bonuses" might prove useful not only to establish rearrangement distances, but also NP-hardness proofs. For this reason it has become one of my first approach when tackling a new rearrangement problem.

I will add that this dual layered view might also be seen as a mere first step. One might put additional layers in the description of the bonuses themselves as an attempt to better understand how a problem works, or to better express a NP-hard aspect. For example, staying on the genome halving example, I could state that 2-cycles and 1-paths already present in the graph are *bonuses of order zero*, while bigger even cycles and odd paths are *bonuses of higher order*, since they will create additional zero order bonuses at a later step in the scenario.

Interestingly enough, when using such further layering with rearrangement problems, we notice that *(number of expected operations) - (sum of zero order bonuses)* is generally exactly the breakpoint distance.

3.2.2 Scenario, distance and complexity class

Note: As it is rather easy to prove, I don't think this result is new, and I admit I did not actively look for such result in bibliography (I would not know where to look as it is very general). However, since I have seen that question being brought up several times without an answer, I am including my quick proof just in case. I would also like to thank Eric Tannier for pointing out this is a result of self-reducibility.

Given an optimal scenario, computing the minimal distance is trivial. It follows that a polynomial scenario implies a polynomial distance. I prove the converse is true as well, and thus that most rearrangement problems are *self-reducible* under reasonable assumptions on the operation model.

To make the proof easier to follow I will assume we are in the DCJ operation model, while

generalization to other models will be discussed afterwards.

Lemma 1 *On a given genome, there is a polynomial number of possible DCJ (with respect to the genome length).*

Proof 1 *A DCJ is defined on a* fixed *number of breakpoints, $b = 2$, independent of the genome length. It follows there are $O(n^b) = O(n^2)$ possible ways of placing them on the genome. Naturally, the fact there are two possible DCJ that can be performed for each breakpoint positionning doesn't matter as we still have $O(n^2)$ possible DCJ on a genome of length n*■

Lemma 2 *Optimal DCJ scenarios have a polynomial length (with respect to the genome length).*

Proof 2 *The breakpoint distance is an upperbound for the DCJ distance and it is a number that is polynomial with respect to the genome length.*■

Theorem 1 *If the distance can be computed in polynomial time, then an optimal scenario can be as well.*

Proof 3 *If an operation decreases the distance, then it is optimal. An algorithm reconstructing an optimal scenario can always be built this way: At each step, we look for an optimal operation by trying all possible breakpoints positions, and compute the distance for the resulting genome. By lemma 1 each step is polynomial iff the distance computation is polynomial, and by lemma 2, there is a polynomial number of iterations. Thus, this algorithm is polynomial on the whole iff the distance computation is polynomial.*■

Naturally, this result can be generalized to other operation models, as long as 1) there is a polynomial number of possible operations at each step, whose resulting genomes can each be computed polynomially and 2) the distance itself is a polynomial number with respect to the genome length.

One can verify this holds for most common operation models such as reversals, translocations, block interchange, transpositions, and also k-breaks.

As a final note, I'll add that the resulting scenario computing algorithm is far from efficient and that its interest is most likely limited to this self-reducibility result and its implications.

3.3 Model I: Breakpoint duplication

This work has been published in [Thomas et al., 2011].

Minor revisions have been made since, giving more information about the orientation cost for reversals, a proof has been rewritten, and vocabulary has been fixed (the algorithm I designed for this problem was a fixed-parameter tractable (FPT) algorithm, thanks to Laurent Bulteau for pointing out this fact).

Since I could develop the dual layered view in section 3.2.1, I also added a new section explaining the DCJ distance formula and complexity for this problem using this tool.

I proved the genome dedoubling problem is NP-hard for DCJ and reversals, and gave a FPT algorithm in the number of cycles to solve it under DCJ, and another one under reversals for a subclass of genomes.

3.3.1 Biological motivation

Gene duplication is an important source of variations in eukaryotes. Recently, several studies have highlighted biological evidence for abundant segmental duplications that occur around breakpoints of rearrangement events in mammalians [Bailey et al., 2004, Howarth et al., 2011], and in Drosophila species group [Ranz et al., 2007] [Matzkin et al., 2005] [Richards et al., 2005] [Meisel, 2009].

You might refer to these papers or to [Thomas et al., 2011] for more details.

3.3.2 Model

3.3.2.1 Considered genomes

The genomes considered in this section are duplicated and totally duplicated genomes, as defined in section 2.4.

I also introduce a particular kind of duplicated genome, namely *dedoubled genomes.*

Definition 3 *A* dedoubled genome *is a duplicated genome G such that for any duplicated marker x in G, either $(x \ \ \overline{x})$, or $(\overline{x} \ \ x)$ is an adjacency of G.*

For example, $G = (\circ \ \ -1 \ -\overline{1} \ 2 \ \circ) \ (\ 4 \ \overline{4} \ \overline{3} \ 3 \ )$ is a dedoubled genome with 3 duplicated markers.

3.3.2.2 Considered operations

I use two operation models in this section: DCJ, and reversals.

I recall that reversals are particular kinds of DCJ.

I also define new operations as follows:

A *1-breakpoint-duplication DCJ* (1-BD-DCJ) operation on a genome G is a rearrangement operation that alters two different adjacencies $(a \ \ b)$ and $(c \ \ d)$ of G, by:

- first adding marker $\overline{a}$ at the appropriate position to produce segment $(a \ \ \overline{a} \ \ b)$,
- then applying a DCJ operation that cuts adjacencies $(a \ \ \overline{a})$ and $(c \ \ d)$ to produce either $(a \ \ d)$ and $(c \ \ \overline{a})$, or $(a \ \ -c)$ and $(-\overline{a} \ \ d)$.

A *2-breakpoint-duplication DCJ* (2-BD-DCJ) operation on a genome G is a rearrangement operation that alters two different adjacencies $(a \ \ b)$ and $(c \ \ d)$ of G, by:

- first adding markers $\overline{a}$ and $\overline{c}$ at the appropriate positions to produce segments $(a \ \ \overline{a} \ \ b)$ and $(c \ \ \overline{c} \ \ d)$,
- then applying a DCJ operation that cuts adjacencies $(a \ \ \overline{a})$ and $(c \ \ \overline{c})$ to produce either $(a \ \ \overline{c})$ and $(c \ \ \overline{a})$, or $(a \ \ -c)$ and $(-\overline{a} \ \ \overline{c})$.

In this context, a regular DCJ could also be regarded as a 0-BD-DCJ. I will include this possibility in the following general definition of a BD-DCJ, since regular DCJ can also be used in BD-DCJ scenarios.

Definition 4 *A* breakpoint-duplication DCJ *(BD-DCJ) operation on a genome G is either a DCJ, a 1-BD-DCJ operation, or a 2-BD-DCJ operation.*

In the sequel, if some markers are duplicated by a BD-DCJ operation, they are indicated in bold font in the initial genome. For example, the following rearrangement is a 2-BD-DCJ operation that acts on adjacencies $(-2\ \ -1)$ and $(4\ \ -3)$, and duplicates markers 2 and 4. The intermediate step resulting in the duplication of markers 2 and 4 is shown above the arrow.

$$(1 \blacktriangle \mathbf{2})\ (\circ\ \ 3\ \ \blacktriangle\ \mathbf{-4}\ \ \circ) \overset{(1\ \cdot\ \overline{2}\ \blacktriangle\ 2)\ (\circ\ \ 3\ \cdot\ -\overline{4}\ \blacktriangle -4\ \ \circ)}{\rightarrow} (\circ\ \ 3\ \cdot\ -\overline{4}\ \cdot\ 2\ \ 1\ \cdot\ \overline{2}\ \cdot\ -4\ \ \circ)$$

To summarize, a BD-DCJ operation consists of a *first step* in which zero, one or two markers are duplicated, followed by a *second step* where a DCJ operation is applied. Similarly, we now define a *breakpoint-duplication reversal* (BD-reversal) operation.

Definition 5 *A* breakpoint-duplication reversal *(BD-reversal) operation on a genome G is a BD-DCJ operation such that the DCJ operation applied in the second step of the BD-DCJ operation is a reversal.*

For example, the following rearrangement is a BD-reversal that is a 1-BD-DCJ operation that acts on adjacencies $(2\ \ -1)$ and $(-3\ \ 4)$, and duplicates marker 2.

$$(\circ\ \ 1 \blacktriangle \mathbf{-2}\ \ -3 \blacktriangle 4\ \ \circ) \overset{(\circ\ \ 1\ \cdot\ -\overline{2}\ \blacktriangle\ -2\ -3\ \blacktriangle\ 4\ \ \circ)}{\rightarrow} (\circ\ \ 1\ \cdot\ -\overline{2}\ \cdot\ 3\ \ 2\ \cdot\ 4\ \ \circ)$$

A *BD-DCJ scenario* (resp. *BD-reversal scenario*) between a non-duplicated genome A and a duplicated genome B is a sequence composed of BD-DCJ (resp. BD-reversal) operations allowing to transform A into B.

Definition 6 *Given a non-duplicated genome A and a duplicated genome B, the* BD-DCJ distance *(resp.* BD-reversal distance*) between A and B is the minimal length of a BD-DCJ (resp. BD-reversal) scenario between A and B.*

We now give an obvious, but useful property allowing to reduce a BD-DCJ scenario to a DCJ scenario.

Proposition 1 *Given a non-duplicated genome A and a duplicated genome B, for any a BD-DCJ (resp. BD-reversal) scenario between A and B, there exists a DCJ (resp. reversal) scenario of same length between a dedoubled genome D and B such that the reduction of D is A ($D^R = A$).*

Proof 4 *Let S be a BD-DCJ (resp. BD-reversal) scenario between A and B. D is the genome obtained from A, by adding, for any marker x duplicated by a BD-DCJ operation in S, the marker $\overline{x}$ in a way to produce either adjacency $(\overline{x}\ x)$, or $(x\ \overline{x})$ as done in S.*

It is easy to see that $D^R = A$. The DCJ (resp. reversal) scenario between D^R and B having the same length as S, is the sequence of DCJ (resp. reversal) contained in S or in BD-DCJ (resp. BD-reversal) operations of S, with the same order as in S.■

For example, in the following, a BD-reversal scenario between $A = (\circ\ 1\ \ 2\ \ 3\ \ 4\ \ 5\ \circ)$ and $B = (\circ\ 1\ -\overline{4}\ \ 2\ \ \overline{3}\ -\overline{5}\ -\overline{2}\ -\overline{1}\ \ 4\ -3\ \ 5\ \circ)$ is transformed into a reversal scenario between $D = (\circ\ 1\ \ \overline{1}\ \ \overline{2}\ \ 2\ \ \overline{3}\ \ 3\ \ \overline{4}\ \ 4\ \ \overline{5}\ \ 5\ \circ)$ and B.

BD-reversal scenario	Reversal scenario
$A = (\circ\ \mathbf{1}\ \blacktriangle\ 2\ \ 3\ \blacktriangle\ \mathbf{4}\ \ 5\ \circ)$	$D = (\circ\ 1\ \blacktriangle\ \overline{1}\ \ \overline{2}\ \ 2\ \ \overline{3}\ \ 3\ \ \overline{4}\ \blacktriangle\ 4\ \ 5\ \ \overline{5}\ \circ)$
$(\circ\ 1\ -\overline{4}\ \blacktriangle\ -3\ -\mathbf{2}\ \blacktriangle\ -\overline{1}\ \ 4\ \ 5\ \circ)$	$(\circ\ 1\ -\overline{4}\ \blacktriangle\ -3\ -\overline{3}\ -2\ \blacktriangle\ -\overline{2}\ -\overline{1}\ \ 4\ \ 5\ \ \overline{5}\ \circ)$
$(\circ\ 1\ -\overline{4}\ \ 2\ \ 3\ \blacktriangle\ -\overline{2}\ -\overline{1}\ \ 4\ \blacktriangle\ 5\ \circ)$	$(\circ\ 1\ -\overline{4}\ \ 2\ \ \overline{3}\ \ 3\ \blacktriangle\ -\overline{2}\ -\overline{1}\ \ 4\ \blacktriangle\ \ 5\ \ \overline{5}\ \circ)$
$(\circ\ 1\ -\overline{4}\ \ 2\ \blacktriangle\ \mathbf{3}\ -4\ \ \overline{1}\ \ \overline{2}\ \ \mathbf{5}\ \blacktriangle\ \circ)$	$(\circ\ 1\ -\overline{4}\ \ 2\ \ \overline{3}\ \blacktriangle\ 3\ -4\ \ \overline{1}\ \ \overline{2}\ 5\ \blacktriangle\ \overline{5}\ \circ)$
$(\circ\ 1\ -\overline{4}\ \ 2\ \ \overline{3}\ -5\ -\overline{2}\ -\overline{1}\ \ 4\ -3\ \ \overline{5}\ \circ)$	$(\circ\ 1\ -\overline{4}\ \ 2\ \ \overline{3}\ -5\ -\overline{2}\ -\overline{1}\ \ 4\ -3\ \ \overline{5}\ \circ)$

3.3.3 Genome Dedoubling

I now state the genome dedoubling problem.

Definition 7 *Given a duplicated genome G, the* DCJ (resp. reversal) genome dedoubling problem *consists in finding a non-duplicated genome H such that the BD-DCJ (resp. BD-reversal) distance between H and G is minimal.*

Given a duplicated genome G, we denote by $d_{dcj}(G)$ (resp. $d_{rev}(G)$), the minimum BD-DCJ (resp. BD-reversal) distance between any non-duplicated genome and G. From Proposition 1, the following proposition is straigthforward.

Proposition 2 *Given a duplicated genome G, the DCJ (resp. reversal) genome dedoubling problem on G is equivalent to finding a dedoubled genome D such that the DCJ (resp. reversal) distance between D and G is minimal.*

The next proposition describes a further reduction of the genome dedoubling problem on a duplicated genome G.

Proposition 3 *Given a duplicated genome G, the* DCJ (resp. reversal) genome dedoubling problem *on G is equivalent to the* DCJ (resp. reversal) genome dedoubling problem *on the totally duplicated genome G^T obtained from G by replacing every maximal subsequence of non-duplicated markers beginning with a marker x by the pair $x\ \overline{x}$.*

Proof 5 *By definition, building adjacencies of the type $(x\ \overline{x})$ or $(\overline{x}\ x)$is the focus of breakpoint-duplication rearrangement scenarios, and as it will be shown in the next sections, destroying such already formed adjacencies is never needed. Therefore, there exists an optimal scenario that preserves the consecutivity of unduplicated markers grouped into a subsequence.*■

For example, solving the DCJ (resp. reversal) genome dedoubling problem on

$G = (\circ\ 1\ \ 4\ -\mathbf{7}\ \ \overline{1}\ -\mathbf{5}\ \ \mathbf{10}\ -8\ -\overline{4}\ \ \mathbf{2}\ \ \mathbf{6}\ -\mathbf{9}\ \ \mathbf{3}\ \ -\overline{8}\ \circ)$ is equivalent to solving it on

$G^T = (\circ\ 1\ \ 4\ -\mathbf{7}\ -\overline{\mathbf{7}}\ \ \overline{1}\ -\mathbf{5}\ -\overline{\mathbf{5}}\ -8\ -\overline{4}\ \ \mathbf{2}\ \ \overline{\mathbf{2}}\ \ -\overline{8}\ \circ)$.

The transformations applied on G to obtain G^T are indicated in bold font.

In the sequel, G will always denote a totally duplicated genome, and we focus in Sections 3.3.4 and 3.3.5 on the problem of finding a dedoubled genome D such that the DCJ (resp. reversal) distance between D and G is minimal.

3.3.4 DCJ

In this section, G denotes a totally duplicated genome. In order to give a formula for the DCJ dedoubling distance of G, $d_{dcj}(G)$, we use a graph called the *dedoubled adjacency graph* of G.

3.3.4.1 Dedoubled adjacency graph

Definition 8 *The* dedoubled adjacency graph *of G, denoted by $\mathcal{A}(G)$, is the graph whose vertices are the adjacencies of G, and for any marker x there is one edge between the vertices $(x\ u)$ and $(v\ \overline{x})$, and one edge between the vertices $(y\ x)$ and $(\overline{x}\ z)$.*

An example of dedoubled adjacency graph is depicted in Fig. 3.2. In the following, we will simply refer to dedoubled adjacency graphs as adjacency graphs.

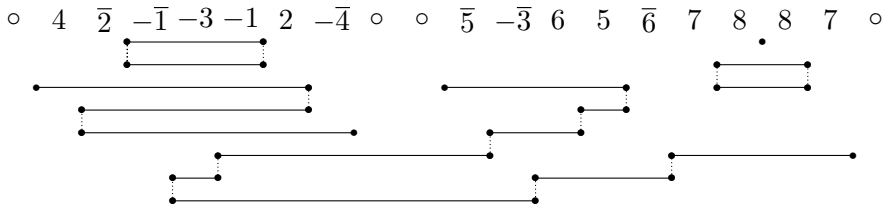


Figure 3.2: The adjacency graph of $G = (\circ\ 4\ \ \overline{2}\ -\overline{1}\ -3\ -1\ \ 2\ -\overline{4}\ \circ)\ \ (\circ\ \overline{5}\ -\overline{3}\ \ 6\ \ 5\ \ \overline{6}\ \ 7\ \ 8\ \ 8\ \ 7\circ)$

Note that all vertices in $\mathcal{A}(G)$ have degree one or two. Thus, the connected components of $\mathcal{A}(G)$ are only paths and cycles. These paths and cycles are called *elements* of $\mathcal{A}(G)$.

Given a couple of paralogous markers $(x, \overline{x})$, an element of the graph $\mathcal{A}(G)$ is said to *contain* the couple $(x, \overline{x})$ if it contains the edge linking vertices $(x\ u)$ and $(v\ \overline{x})$, or the edge linking vertices $(y\ x)$ and $(\overline{x}\ z)$.

By definition, a couple $(x, \overline{x})$ can possibly be contained in only one element A of $\mathcal{A}(G)$ if said element A contains both edges $((x\ u), (v\ \overline{x}))$ and $((y\ x), (\overline{x}\ z))$. In this case, A is said to *contain twice* the couple $(x, \overline{x})$, and A is called a *duplicated* element of $\mathcal{A}(G)$. If an element A contains no couple $(x, \overline{x})$ twice, then it is called a *non-duplicated* element of $\mathcal{A}(G)$. If the two edges $((x\ u), (v\ \overline{x}))$ and $((y\ x), (\overline{x}\ z))$ belong to two different elements A and B of $\mathcal{A}(G)$, then A and B will both contain $(x, \overline{x})$. In this case, we say that A and B *intersect*. If two elements A and B do not intersect, then we say that A and B are *independent*. For example in Fig. 3.2 the two paths of the adjacency graph are duplicated, while the three cycles are non-duplicated. The leftmost path and the leftmost cycle intersect because they both contain the couple $(2, \overline{2})$, while the two paths are independent.

Given an element A of $\mathcal{A}(G)$, the *set induced by* A is the set of couples $(x, \overline{x})$ contained in A.

3.3.4.2 General sorting

In this section, I prove the following theorem:

Theorem 2 *Let n be the number of couples of paralogous markers in G. Let C_i be the maximum size of a subset of non-duplicated pairwise independent cycles in $\mathcal{A}(G)$. The DCJ dedoubling distance of G is $d_{dcj}(G) = n - C_i$. It is NP-hard to compute.*

For example, in Fig. 3.2, the maximum size of a subset of non-duplicated pairwise independent cycles is 2 as there are three cycles, and the two rightmost cycles intersect. The distance would then be $d_{dcj}(G) = 8 - 2 = 6$.

To prove Theorem 2, I use the following properties:

Property 1 *Let n be the number of couples of paralogous markers in G.*

1. *The maximum size C_i of a set of pairwise independent cycles in the graph $\mathcal{A}(G)$ is n, since each of the n couples $(x, \overline{x})$ is contained in at most one cycle of a set of pairwise independent cycles.*

2. *By definition of a dedoubled genome, if G is a dedoubled genome, then the graph $\mathcal{A}(G)$ has n non-duplicated pairwise independent cycles, each one containing a single couple of paralogous markers, plus possibly other cycles. Thus, in this case, $C_i = n$.*

3. *A DCJ operation can only alter the maximum size C_i of a set of pairwise independent cycles by -1, 0 or $+1$, because a DCJ operation can only either extract a new cycle that contributes to increase C_i by 1, or destroy a single cycle in any set of pairwise independent cycles, thus decreasing C_i by 1, or leaves C_i unchanged.*

Algorithm 1 is an algorithm that provides a $n - C_i$ length DCJ scenario transforming G into a dedoubled genome. It is FPT in the number of cycles in the graph.

Algorithm 1 Transforming a totally duplicated genome G into a dedoubled genome by DCJ

```
Construct 𝒜(G).
Choose a maximum size set S_i of non-duplicated pairwise independant cycles.
for Any couple (x, x̄) of paralogous markers do
    if (x, x̄) is contained in a cycle c of S_i containing more than one couple then
        Perform the DCJ that creates adjacency (x x̄) or (x̄ x) by splitting c
        into two cycles, one of the cycles containing only the couple (x, x̄).
        Replace c in S_i by the two new cycles.
    else
        Perform any DCJ that creates adjacency (x x̄) or (x̄ x), unless such adjacency is already
        present.
    end if
end for
```

We now have all the pre-requisites to give the proof of Theorem 2.

Proof 6 *of Theorem 2. From Property 1, a DCJ operation cannot increase C_i by more than 1. Algorithm 1 provides a DCJ scenario transforming G into a dedoubled genome, by increasing C_i by 1 at each DCJ operation until it reaches its upper bound n. Algorithm 1 then provides a minimum length scenario which is of length $n - C_i$ (any shorter scenario would necessarily increase C_i by more than 1 at some point which is a contradiction).*■

Lemma 3 *Choosing a maximum size set of pairwise independant cycles is a NP-hard, APX-complete problem, approximable with an approximation ratio of 2.*

Proof 7 *We show the equivalence of the problem with a* 2-frequency Maximum Set Packing, *known to be APX-complete [Berman and Fujito, 1995] and 2-approximable [Hochbaum, 2004]. A* 2-frequency collection of sets *is a collection of finite sets such that each element of any set belongs to at most two sets of the collection. Given a 2-frequency collection C_n of sets, the 2-frequency Maximum Set Packing problem on C_n asks to find the maximum number of pairwise disjoint sets in C_n.*

Computing the maximum size C_i of a subset of non-duplicated pairwise independent cycles in $\mathcal{A}(G)$ can obviously be reduced to the 2-frequency Maximum Set Packing problem *on a 2-frequency collection C_n of sets:*

- *Treat each non-duplicated cycle in $\mathcal{A}(G)$ as the set of its edges.*
- *Transform them into sets from C_n by replacing each edge $(x\ \overline{x})$ or $(\overline{x}\ x)$ by the element x.*

Conversely, a 2-frequency collection C_n of sets, containing elements in the form $(k_1, ..., k_n)$, can be converted into a totally duplicated genome G such that the non-duplicated cycles of $\mathcal{A}(G)$ induce the sets of C_n:

- *Create markers i and $\overline{i}$ for all distinct element from all sets.*
- *For each set $(k_1, ..., k_n)$ create all adjacencies $(k_i\ \overline{k_{i+1}})$, as well as $(k_n\ \overline{k_1})$ (if one marker extremity is already taken, use both paralogs instead, creating $(\overline{k_i}\ k_{i+1})$ or $(\overline{k_n}\ k_1)$).*
- *Put telomeres on all remaining free extremities.*

■

Corollary 1 *The Genome Dedoubling problem by DCJ is NP-complete. Algorithm 1 solves the problem in linear time complexity, except for the computation of the set of cycles S_i that is 2-approximable.*

Corollary 2 *As the computation of the set of cycles S_i can be done in exponential time with respect to the total number of cycles, algorithm 1 is FPT in the number of cycles in $\mathcal{A}(G)$. Indeed, the number of cycles in the graph is independent from the genome length (since it is always possible to merge cycles with DCJ, it is possible to build arbitrarily long genomes with any fixed number of cycles).*

3.3.4.3 Using the dual layered view

In this section, I will use the ideas I developed in section 3.2.1 to quickly establish the distance formula and thus show the same basic reasonings still hold for more complex problems.

First, the *number of expected operations* is n. This is straightforward since we are trying to reconstruct n doublets and a DCJ can always reconstruct one.

Now, in order to establish what is the *sum of bonuses*, we have to keep the graph in mind.

We are trying to reconstruct 1-cycles for each of the markers, and although each marker appears twice in the graph, we only need one of them. This implies the following properties.

1. The bonuses are all in the cycles, since only the cycles will eventually give a 1-cycle as remainder.
2. However, if a cycle contains a same marker twice, it won't count as a bonus. This is because having adjacency $(x\overline{x})$ as free remainder when adjacency $(\overline{x}x)$ is already present is not a bonus, since having both is not necessary.
3. More generally, and for the same reason, if we have a cycle that contains $(x\overline{x})$, then another cycle containing $(\overline{x}x)$ won't count as a bonus either.

It follows that there are as many bonuses as there are "non-duplicated (property 2) independent (property 3) cycles (property 1)".

The minimum distance is reached through the maximum number of bonuses. Therefore we are looking for a maximum number of non-duplicated independent cycles. This is exactly the 2-frequency set-packing problem, which is NP-hard.

3.3.4.4 Sorting between linear unichromosomal genomes

In this section, as a first step to study genome dedoubling by reversal, we search for a minimum length DCJ scenario that transforms G into a dedoubled genome consisting of a single linear chromosome.

In this section and the sequel, G denotes a totally duplicated genome consisting of a single linear chromosome. In this case, the graph $\mathcal{A}(G)$ contains exactly *one path, and possibly several cycles.*

Definition 9 *The path in $\mathcal{A}(G)$ is said to be* valid *if it contains every couple $(x, \overline{x})$ of paralogous markers in G.*

A DCJ operation that merges a cycle c of $\mathcal{A}(G)$ in the path p is a DCJ operation that acts on an adjacency of c and an adjacency of p, thus gathering c and p into a longer path.

Note that if G is a dedoubled genome, then the path in $\mathcal{A}(G)$ is necessarily valid. We call such a genome a *dedoubled linear genome.* So, if the path in $\mathcal{A}(G)$ is not valid, then any DCJ scenario transforming G into a dedoubled linear genome will merge, in the path, cycles containing the couples $(x, \overline{x})$ that are not contained in the path.

In the following, we always denote by m the minimum number of cycles required to make the path of $\mathcal{A}(G)$ valid. We also denote by C_i the maximum size of a subset of non-duplicated pairwise independent cycles. First, we have the following property:

Property 2 *Let C be the number of cycles in $\mathcal{A}(G)$. We have $C_i = C - m$.*

Proof 8 *Let $\mathcal{S}$ be the set of all cycles from G ($|\mathcal{S}| = C$). If $\mathcal{S}_m$ is a set of cycles that can be merged in the path to make it valid, then necessarily $\mathcal{S} - \mathcal{S}_m$ is a set of non-duplicated pairwise independent cycles:*

- *Any cycle c contained in $\mathcal{S}-\mathcal{S}_m$ is non-duplicated, otherwise the path obtained after merging would not be valid as it would not contain any couple of paralogous markers $(x, \overline{x})$ contained twice in c.*

- *Any two cycles c_1 and c_2 of $\mathcal{S} - \mathcal{S}_m$ are independent, otherwise the path obtained after merging would not be valid as it would not contain any couple of paralogous markers $(x, \overline{x})$ contained in both c_1 and c_2.*

■

From Property 2, we then have the following lemma.

Lemma 4 *Let n be the number of couples of paralogous markers in G. Let C be the number of cycles in $\mathcal{A}(G)$. The minimum length d of a DCJ scenario transforming G into a dedoubled genome consisting of a single linear chromosome equals $d = n - C + 2m$.*

Proof 9 *First, from Property 2, we have $n - C + 2m = n - C_i + m$. Similarly to C_i, a DCJ operation can only alter m by $+1$, -1 or 0. Next, a DCJ operation that increases C_i by 1 also increases C by 1, and then leaves m unchanged. Conversely, a DCJ operation that decreases m by 1 leaves C_i unchanged.*

Algorithm 1 in which we add the line (2': Merge in the path all the cycles that are not in S_i) between lines 2 and 3 provides a DCJ scenario that first decreases m until it reaches its lower bound 0 (in m DCJ operations), then increases C_i until it reaches its upper bound n (in $n - C_i$ DCJ operations). ■

Corollary 3 *The problem of finding a DCJ scenario transforming G into a dedoubled genome consisting of a single linear chromosome is NP-hard. Algorithm 1, in which we add the line (2': Merge in the path all the cycles that are not in S_i) between lines 2 and 3, solves the problem in linear time complexity, except for the computation of the set of cycles S_i that is 2-approximable.*

3.3.5 Reversal

The graph used in this section behaves like the *arc overlap graph* used in [Bergeron, 2005] for the Hannenhalli-Pevzner theory of sorting by reversal [Hannenhalli and Pevzner, 1995c]. The genome G consists of a single linear chromosome.

3.3.5.1 Dedoubled overlap graph

For any couple $(x, \overline{x})$ of paralogous markers in G, the segment of $(x, \overline{x})$ is the smallest segment of G containing both markers x and $\overline{x}$. For example, in genome $G = (\circ\ 1\ \ 3\ \ \overline{1}\ -\overline{2}\ -4\ -\overline{3}\ \ 2\ -\overline{4}\ \circ)$, the segment of $(1, \overline{1})$ is $(1\ \ 3\ \ \overline{1})$, and the segment of $(2, \overline{2})$ is $(-\overline{2}\ -4\ -\overline{3}\ \ 2)$.

Definition 10 *The* dedoubled overlap graph *of G, denoted by $\mathcal{O}(G)$, is the graph whose vertices are all the couples $(x, \overline{x})$ of paralogous markers of G, and there is an edge between two vertices $(u, \overline{u})$ and $(v, \overline{v})$ if the segments of u and v overlap.*

An example of dedoubled overlap graph is depicted in Fig. 3.3.a. In the following, I will simply refer to dedoubled overlap graphs as overlap graphs.

The vertex $(x, \overline{x})$ of the graph $\mathcal{O}(G)$ is *oriented* if x and $\overline{x}$ have different signs in G, otherwise it is *unoriented*. If the vertex $(x, \overline{x})$ is oriented then there exists a reversal operation denoted

by Rev($x\ \ \overline{x}$) that produces the adjacency ($x\ \ \overline{x}$) and a reversal operation denoted by Rev($\overline{x}\ \ x$) that produces the adjacency ($\overline{x}\ \ x$). For example, in genome $G = (\circ\ 1\ \ 3\ \ \overline{1}\ -\overline{2}\ -4\ -\overline{3}\ \ 2\ -\overline{4}\ \circ)$, both copies of 3 have opposite sign, therefore $(3, \overline{3})$ is an oriented vertex of $\mathcal{O}(G)$.

Rev($3\ \ \overline{3}$) = $(\circ\ 1\ \ 3\ \ \underline{\overline{1}\ -\overline{2}\ -4\ -\overline{3}}\ \ 2\ -\overline{4}\ \circ) \rightarrow (\circ\ 1\ \ 3\ \ \overline{3}\ \ 4\ \ \overline{2}\ -\overline{1}\ \ 2\ -\overline{4}\ \circ)$.

Rev($\overline{3}\ \ 3$) = $(\circ\ 1\ \ \underline{3\ \ \overline{1}\ -\overline{2}\ -4}\ -\overline{3}\ \ 2\ -\overline{4}\ \circ) \rightarrow (\circ\ 1\ \ 4\ \ \overline{2}\ -\overline{1}\ -3\ \ -\overline{3}\ \ 2\ -\overline{4}\ \circ)$.

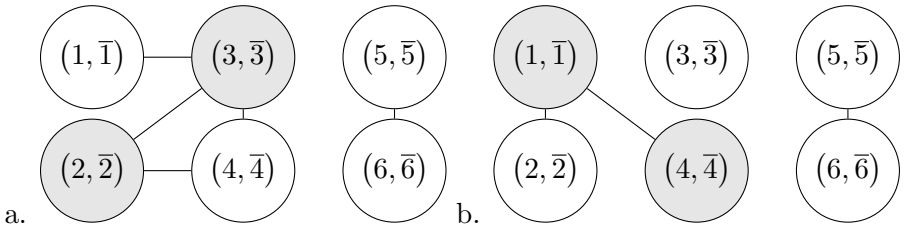


Figure 3.3: a. The overlap graph of $G = (\circ\ 1\ \ 3\ \ \overline{1}\ -\overline{2}\ -4\ -\overline{3}\ \ 2\ -\overline{4}\ \circ)\ \ (\circ\ \overline{5}\ 6\ \ 5\ \ \overline{6}\ \circ)$. Oriented vertices are colored in grey. The graph $\mathcal{O}(G)$ has two connected components, one oriented and one unoriented. b. the overlap graph obtained after applying the reversal Rev($3\ \ \overline{3}$) to produce adjacency ($3\ \ \overline{3}$).

The overlap graph of G behaves like arc overlap graphs used in [Bergeron, 2005] for the Hannenhali-Pevzner theory of sorting by reversal [Hannenhalli and Pevzner, 1995c]. Indeed, given an oriented vertex $(x, \overline{x})$ of the graph $\mathcal{O}(G)$, performing the reversal Rev($x\ \ \overline{x}$) or Rev($\overline{x}\ \ x$) complements the subgraph induced by $(x, \overline{x})$ and all its neighbouring vertices, and changes the orientation of all vertices in this subgraph (see Fig. 3.3.b).

A connected component of the graph $\mathcal{O}(G)$ is *oriented* if it contains at least one oriented vertex, otherwise it is *unoriented*. A genome is *oriented* if all connected components of the graph $\mathcal{O}(G)$ are oriented, otherwise it is *unoriented*.

Given an oriented vertex $(x, \overline{x})$ of the graph $\mathcal{O}(G)$, the score of $(x, \overline{x})$ is the number of oriented vertices in the genome obtained after applying Rev($x\ \ \overline{x}$) on G. Note that the same number of oriented vertices is obtained after applying Rev($\overline{x}\ \ x$) on G.

If you are familiar with other papers on sorting by reversal, you might notice this score definition differs from the usual one. In fact they both produce the same ranking of vertices.

Property 3 *Let $(x, \overline{x})$ be an oriented vertex of $\mathcal{O}(G)$ of maximum score. Performing Rev($x\ \ \overline{x}$) or Rev($\overline{x}\ \ x$) does not create new unoriented connected components in the overlap graph of the genome obtained.*

Proof 10 *Because the vertices ranking is the same, the proof goes the same way as the proof of Theorem 10 in [Bergeron, 2005]: if Rev($x\ \ \overline{x}$) or Rev($\overline{x}\ \ x$) creates a new unoriented connected component C in the overlap graph. Then, C necessarily contains a vertex $(w, \overline{w})$ that was adjacent to $(x, \overline{x})$ and oriented before applying Rev($x\ \ \overline{x}$) or Rev($\overline{x}\ \ x$), and such that the score of $(w, \overline{w})$ was greater than the score of $(x, \overline{x})$, which is a contradiction.*■

In the sequel, we focus on sorting oriented genomes using reversal dedoubling scenarios. A totally duplicated genome G consisting of a single linear chromosome is called a *valid-path genome* if the single path in $\mathcal{A}(G)$ is valid. Otherwise, it is called a *non-valid-path genome*.

3.3.5.2 Sorting an oriented valid-path genome

In this section, we consider an oriented valid-path genome G. With n the number of couples of paralogous markers in G, and C the number of cycles in $\mathcal{A}(G)$, I prove the following theorem:

Theorem 3 *The reversal dedoubling distance of G is $d_{rev}(G) = n - C$.*

Proof 11 *In Algorithm 2, since G is a valid path genome, then S_i is the set of all cycles. Thus, Algorithm 2 provides a reversal scenario of length $n - C$, which is the smallest length that can be reached since $d_{dcj}(G) = n - C_i = n - C + m = n - C$ as $m = 0$ here, and a $d_{dcj}(G) \leq d_{rev}(G)$ since a reversal scenario is also a DCJ scenario.*■

Algorithm 2 Transforming an oriented genome G into a dedoubled genome by reversals

1: Construct $\mathcal{A}(G)$.
2: Construct $\mathcal{O}(G)$.
3: Choose a maximum size set S_i of non-duplicated pairwise independant cycles.
4: Merge in the path all the cycles that are not in S_i.
5: **while** G is not a dedoubled genome **do**
6: Pick a maximum score vertex $(x, \overline{x})$ in $\mathcal{O}(G)$.
7: **if** $(x, \overline{x})$ is contained in a cycle c of S_i **then**
8: Choose between Rev(x $\overline{x}$) and Rev($\overline{x}$ x) the reversal that splits c into two cycles, and perform it.
9: Replace c in S_i by the two new cycles.
10: Apply the modification induced by the reversal on $\mathcal{O}(G)$.
11: **else**
12: Perform any reversal that creates adjacency (x $\overline{x}$) or ($\overline{x}$ x).
13: **end if**
14: **end while**

3.3.5.3 Sorting an oriented non-valid-path genome

In this section, G denotes an oriented non-valid path genome. At least m cycles of $\mathcal{A}(G)$ have to be merged in the path to make it valid.

An edge $((x\ u), (v\ \overline{x}))$ or $((y\ x), (\overline{x}\ z))$ of the adjacency graph $\mathcal{A}(G)$ is called oriented if markers x and $\overline{x}$ have different signs. Note that extracting a cycle from any element of the graph $\mathcal{A}(G)$ requires this element to contain oriented edges. It is easy to see that given two adjacencies picked in a given element, a reversal acting on these adjacencies will extract a cycle if and only if the path linking these adjacencies contains an odd number of oriented edges. Thus, we have the following lemma:

Lemma 5 *Merging a cycle in the path never creates unoriented connected components in $\mathcal{O}(G)$.*

Proof 12 *A* component *of G is the smallest segment of G containing all markers of a connected component in $\mathcal{O}(G)$. Breakpoints of merging reversals can always be inside two distinct components of the genome. Performing the operation gathers these two components of the genome, possibly with some other as well, into a single component, obviously containing all couples of*

paralogous markers that used to be in the merged components. Therefore, it has to be an oriented component. Assuming otherwise, it would be possible to extract a cycle from an unoriented component, which is nonsense, by performing the inverse operation in the resulting genome.■

Theorem 4 *Let G be a non-valid-path oriented genome. Let C be the number of cycles in the graph and m be the minimum number of cycles to merge in the path to make it valid. The reversal dedoubling distance of G is $d_{rev}(G) = n - C + 2m$.*

Proof 13 *From lemma 4, we have that $d_{rev}(G) \geq n - C + 2m$ as a reversal scenario is always a DCJ scenario. Algorithm 2 provides a scenario of length $n - C_i + m = n - C + 2m$.* ■

Corollary 4 *The Genome Dedoubling problem by reversal on oriented genomes is NP-hard. Algorithm 2 solves the problem in quadratic time complexity, except for the computation of S_i that is 2-approximable.*

3.3.5.4 A few words on unoriented genomes

I reviewed this work with Laurent Bulteau, visiting my lab for a week in May 2012.

We spent time on the last unsolved step: sorting unoriented genomes by reversal.

In [Bergeron et al., 2004], the distance for sorting between non-duplicated genomes by reversal is expressed as $d_{rev} = d_{dcj} + t$, this last term being a parameter called *orientation cost*.

Unfortunately, the techniques used in this article cannot be directly applied to the genome dedoubling problem, leaving open the question of computing this cost.

The work was promising thanks to Laurent Bulteau contribution and it seemed we found a dynamic programming algorithm for computing the orientation cost in polynomial time. However we did not have time to finish this work.

I leave the result here as a conjecture.

Conjecture 1 *The orientation cost for genome dedoubling by reversal can be computed in polynomial time using dynamic programming. This leads to another FPT algorithm in the number of cycles in $\mathcal{A}(G)$ for genome dedoubling by reversal.*

3.3.6 Closing words and credits

Upon presenting this work at *RECOMB-CG 2011, Galway, Ireland*, I learnt from Anne Bergeron that the DCJ genome dedoubling was a problem Aida Ouangraoua originally worked on with her, and that she did not know Aida O. kept working on it with other people after she went back to France.

Aida O. explained the dedoubling problem, showed me the reduction from BD scenario to dedoubling scenario, and described the dedoubled adjacency graph but could not prove the DCJ distance.

I took care of proving the DCJ distance, providing a sorting algorithm, as well as studying the problem under reversals (using [Bergeron et al., 2004] as a starting point, I designed a study for the dedoubling problem).

While the original publication contained a biological application using genomes taken from [Ranz et al., 2007] by Aida O., I chose not to include it here, since I have always considered it was a display of circular reasoning[8].

Jean-Stéphane Varré drew some of the figures for the article.

A while later I briefly revisited this work with Laurent Bulteau which led to the aforementioned additions.

I took care of proving every single result from these papers.

3.4 Model II: Whole tandem duplication

I studied this model separately for block interchange and for DCJ.

The block interchange study has been published in [Thomas et al., 2012a], then further extended in a journal version [Thomas et al., 2013].

The DCJ study has been published in [Thomas et al., 2012b].

I proved the single tandem halving problem is polynomial for both BI and DCJ, providing $O(n^2)$ algorithms for the scenario and a $O(n)$ computation for the distance.

3.4.1 Biological motivation

This problem was designed as a starting point on rearrangement problems using segmental duplications as the cause to duplicated content. Since the genome halving is the main polynomial result on duplicated genome rearrangements, I used it as a base and made a tandem duplication variant.

While a tandem duplication of the whole content of a unilinear genome might not make much sense biologically, we were hoping that the results would allow a better insight on more realistic tandem duplication models, as shown in further studies from section 3.5.

3.4.2 Model

3.4.2.1 Considered genomes

The genomes considered in this section are totally duplicated genomes, and perfectly duplicated genomes, as defined in section 2.4.

I also introduce single tandem duplicated genomes (or 1-tandem duplicated genomes), and a process called reduction whose purpose is just to help shortening the definition:

Reduction is the process of rewriting a consecutive sequence of double-adjacencies a single marker. In the following example, genome G could be reduced by rewriting $2 \diamond 4 \diamond 5$ and their paralogs as 10 and $\overline{10}$:

$$G = (\circ \ 1 \ \overline{1} \ 3 \ 2 \diamond 4 \diamond 5 \ 6 \ \overline{6} \ 7 \ \overline{3} \ 8 \ \overline{2} \diamond \overline{4} \diamond \overline{5} \ 9 \ \overline{8} \ \overline{7} \ \overline{9} \ \circ)$$

$$G^r = (\circ \ 1 \ \overline{1} \ 3 \ 10 \ 6 \ \overline{6} \ 7 \ \overline{3} \ 8 \ \overline{10} \ 9 \ \overline{8} \ \overline{7} \ \overline{9} \ \circ)$$

[8] I recall that the breakpoint duplication model was made based on Ranz' observations. Using his own data to validate it is irrelevant.

Definition 11 *A* single tandem duplicated genome *(or* 1-tandem duplicated genome*) is a totally duplicated genome which can be reduced to a genome of the form* $(\circ \;\; x \;\; \overline{x} \;\; \circ)$.

In other words, a tandem duplicated genome is composed of a single linear chromosome where all adjacencies, except the two telomeric adjacencies and the central adjacency, are double-adjacencies. For example, the genome $(\circ\ 1 \diamond 2 \diamond 3 \diamond 4\ \overline{1} \diamond \overline{2} \diamond \overline{3} \diamond \overline{4}\ \circ)$ is a tandem-duplicated genome that can be reduced to $(\circ\ 5\ \overline{5}\ \circ)$ by rewriting $1 \diamond 2 \diamond 3 \diamond 4$ and $\overline{1} \diamond \overline{2} \diamond \overline{3} \diamond \overline{4}$ as 5 and $\overline{5}$.

I recall the perfectly duplicated genome definition.

Definition 12 *A* perfectly duplicated genome *is a totally duplicated genome such that all adjacencies are double-adjacencies, none of them in the form* $(x\ -\overline{x})$.

For example, the genome $(1 \;\; 2 \;\; 3 \;\; 4 \;\; \overline{1} \;\; \overline{2} \;\; \overline{3} \;\; \overline{4})$ is a perfectly duplicated genome, while $(\circ \;\; 1 \;\; 2 \;\; -\overline{2} \;\; -1 \;\; \circ)$ is not. It is to note that this definition is equivalent to the one from [Mixtacki, 2008], which slightly differs from the one originally introduced in [Warren and Sankoff, 2008].

From definitions 11 and 12, we get the following property:

Property 4 *In the case of unichromosomal genomes, a perfectly duplicated genome is a single tandem duplicated genome which has been circularized (the perfectly duplicated genome can be reduced to* $(x\ \overline{x})$*, it just lacks telomeres).*

3.4.2.2 Considered operations

I use two separate operation models in this section: Block Interchange, and DCJ.

Here is a useful property linking BI operations to DCJ operations.

Property 5 *A single BI operation on a linear chromosome is equivalent to two DCJ operations: an excision followed by a reintegration.*

Proof 14 *Let* $(\circ \;\; 1 \;\; U \;\; 2 \;\; V \;\; 3 \;\; \circ)$ *be a genome,* U *and* V *the two intervals that are to be swapped by a block interchange operation,* $1\ 2$ *and* 3 *the intervals constituting the rest of the genome (note that each of them may be empty).*

The first DCJ operation is the excision that produces the adjacency $(1 \;\; V)$ *by extracting and circularizing the interval* $[U\ ;\ 2]$*:*

$$(\circ \;\; 1 \blacktriangle U \;\; 2 \blacktriangle V \;\; 3 \;\; \circ) \rightarrow (\circ \;\; 1 \;\; V \;\; 3 \;\; \circ)(U \;\; 2)$$

The second DCJ operation is the integration that produces the adjacency $(U \;\; 3)$ *by reintegrating the circular chromosome* $(U \;\; 2)$ *in the appropriate way:*

$$(\circ \;\; 1 \;\; V \blacktriangle 3 \;\; \circ)(U \;\; 2 \blacktriangle) \rightarrow (\circ \;\; 1 \;\; V \;\; 2 \;\; U \;\; 3 \;\; \circ).$$

3.4.3 Single tandem halving

Because the single tandem halving problem is close to the genome halving problem, I will recall both problems definitions, with adapted notations to avoid ambiguity between both distances.

Definition 13 *Given a unilinear totally duplicated genome G, the* single tandem halving problem *(or 1-tandem halving problem) consists in finding an* optimal *1-tandem duplicated genome H, such that the distance between G and H is minimal. This minimal distance is called the* 1-tandem halving distance*, and is denoted $d^t(G)$.*

Definition 14 ([Mixtacki, 2008]) *Given a totally duplicated genome G, the* DCJ genome halving problem *consists in finding an* optimal *perfectly duplicated genome H, such that the DCJ distance between G and H is minimal. This minimal distance is called the* genome halving distance *and is denoted $d^p(G)$.*

Thus, when the goal is a 1-tandem duplicated genome, the distance is noted d^t, when the goal is a perfectly duplicated genome, it is d^p.

Moreover, I will also recall the operation model as subscript.

For example, using these notations, $d^p_{DCJ}(G)$ is the DCJ genome halving distance as defined in [Mixtacki, 2008], while $d^t_{BI}(G)$ is the BI 1-tandem halving distance I am about to prove in the next section.

3.4.4 Block Interchange

3.4.4.1 Lowerbound

In this section I give a lowerbound on the BI 1-tandem halving distance of a totally duplicated genome. I use a data structure representing the genome called the *natural graph* introduced in [Mixtacki, 2008].

Definition 15 [Mixtacki, 2008] *The natural graph of a totally duplicated genome G, denoted by $\mathtt{NG}(G)$, is the graph whose vertices are the* adjacencies *of G, and for any marker u there is one edge between $(u\ v)$ and $(\overline{u}\ w)$, and one edge between $(x\ u)$ and $(y\ \overline{u})$.*

Note that the number of edges in the natural graph of a genome G containing n distinct markers, each one present in two copies, is always $2n$. Moreover, since every vertex has degree one or two, then the natural graph consists only of cycles and paths. For example, the natural graph of genome $G = (\circ\ \ 1\ \ \overline{2}\ \ \overline{1}\ \ \overline{4}\ \ 3\ \ 4\ \ \overline{3}\ \ 2\ \ \circ)$ is depicted in figure 3.4.

Definition 16 *Given an integer k, a $k-$*cycle *(resp. $k-$*path*) in the natural graph of a totally duplicated genome is a cycle (resp. path) that contains k edges. If k is even, the cycle (resp. path) is called* even*, and* odd *otherwise.*

Based on the natural graph, a formula for the DCJ halving distance was given in [Mixtacki, 2008]. Given a totally duplicated genome G such that the number of even cycles and the number of odd paths in $\mathtt{NG}(G)$ are respectively denoted by EC and OP, the DCJ halving distance of G is:

$$d^p_{DCJ}(G) = n - \mathrm{EC} - \left\lfloor \frac{\mathrm{OP}}{2} \right\rfloor$$

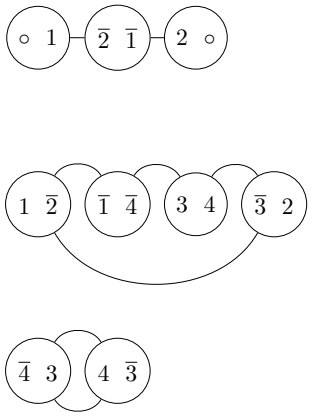


Figure 3.4: The natural graph of genome $G = (\circ\ 1\ \overline{2}\ \overline{1}\ \underline{\overline{4}\ 3}\ \underline{4\ \overline{3}}\ 2\ \circ)$; it is composed of one path and two cycles.

In the case of the BI 1-tandem halving distance, some peculiar properties of the natural graph need to be stated, allowing one to simplify the formula of the DCJ halving distance, and leading to a lowerbound on the BI 1-tandem halving distance.

In the following properties, we assume that G is a genome composed of a single linear chromosome containing n distinct markers, each one present in two copies in G.

Property 6 *The natural graph* $\mathtt{NG}(G)$ *contains only even cycles and paths:*

1. *All cycles in the natural graph* $\mathtt{NG}(G)$ *are even.*
2. *The natural graph* $\mathtt{NG}(G)$ *contains only one path, and this path is even.*

Proof 15 *First, if* $(a\ x)$ *is a vertex of the graph that belongs to a cycle* C*, then there exists an edge between* $(a\ x)$ *and a vertex* $(\overline{a}\ y)$*. These two adjacencies are the only two containing a copy of the marker* a *at the first position. So, if we consider the set of all the first markers in all adjacencies contained in the cycle* C*, then each marker in this set is present exactly twice. Therefore, the cycle* C *is an even cycle.*

Secondly, the graph contains exactly two vertices (adjacencies) containing the marker $\circ$ *which are both necessarily ends of a path in* $\mathtt{NG}(G)$*. Thus there can be only one path in the graph. Since the number of edges in the graph is even and all cycles are even, then the single path is also even.*■

I now give a lowerbound on the minimum length of a DCJ scenario transforming G into a 1-tandem duplicated genome.

Lemma 6 *Let* $d^t_{DCJ}(G)$ *be the minimum DCJ distance between* G *and any 1-tandem duplicated genome. If* $\mathtt{NG}(G)$ *contains* C *cycles then a lowerbound on* $d^t_{DCJ}(G)$ *is given by:*

$$d^t_{DCJ}(G) \geq n - C - 1$$

Proof 16 *First, since all cycles of* $\mathtt{NG}(G)$ *are even and* $\mathtt{NG}(G)$ *contains no odd path, then, from the DCJ halving distance formula, the DCJ halving distance of* G *is* $d^p_{DCJ}(G) = n - C$.

Now, since any 1-tandem duplicated genome can be transformed into a perfectly duplicated genome with one DCJ, then $d^t_{DCJ}+1 \geq d^p_{DCJ}$. *Therefore, we have* $d^t_{DCJ} \geq d^p_{DCJ}-1 \geq n-C-1$.■

We are now ready to state a lowerbound on the BI 1-tandem halving distance of a totally duplicated genome G.

Theorem 5 *If* $\mathtt{NG}(G)$ *contains* C *cycles, then a lowerbound on the BI 1-tandem halving distance is given by:*

$$d^t_{BI}(G) \geq \left\lfloor \frac{n-C}{2} \right\rfloor$$

Proof 17 *We denote by* $\ell(S)$ *the length of a rearrangement scenario* S. *Let* S_{BI} *be a BI scenario transforming* G *into a 1-tandem duplicated genome. From property 5, we have that* S_{BI} *is equivalent to a DCJ scenario* S_{DCJ} *such that* $\ell(S_{DCJ}) = 2*\ell(S_{BI})$. *Now, suppose that* $\ell(S_{BI}) < \lfloor \frac{n-C}{2} \rfloor$, *then* $\ell(S_{BI}) \leq \lfloor \frac{n-C}{2} \rfloor - 1 \leq \lceil \frac{n-C-1}{2} \rceil - 1$.

This implies $\ell(S_{DCJ}) \leq 2\lceil \frac{n-C-1}{2} \rceil - 2 \leq n - C - 2 < n - C - 1$. *Thus, from Lemma 6 we have* $\ell(S_{DCJ}) < d^t_{DCJ}$ *which contradicts the fact that* d^t_{DCJ} *is the minimal number of DCJ operations required to transform* G *into a 1-tandem duplicated genome.*

In conclusion, we always have $d^t_{BI}(G) \geq \lfloor \frac{n-C}{2} \rfloor$. ■

3.4.4.2 Distance

In this section, I show that the established lowerbound is in fact the actual distance.

In other words, the BI 1-tandem halving distance of a totally duplicated genome G with n distinct markers such that $\mathtt{NG}(G)$ contains C cycles is exactly:

$$d^t_{BI}(G) = \left\lfloor \frac{n-C}{2} \right\rfloor$$

Which means that enforcing the constraint that successive couples of consecutive DCJ operations have to be equivalent to BI operations does not change the distance even though it obviously restricts the DCJ that can be performed at each step of the scenario.

In the following, G denotes a totally duplicated genome G constisting in a single linear chromosome with n distinct markers after the reduction process, and such that $\mathtt{NG}(G)$ contains C cycles. We begin by recalling some useful definitions and properties of the DCJ operations that allow one to decrease the DCJ halving distance by 1 in the resulting genome.

Definition 17 *A DCJ operation on* G *producing genome* G' *is* sorting *if it decreases the DCJ halving distance by* 1*:* $d^p_{DCJ}(G') = d^p_{DCJ}(G) - 1 = n - C - 1$.

Since the number of distinct markers in G' is n and $d^p_{DCJ}(G') = n - C - 1$, then $\mathtt{NG}(G')$ contains $C + 1$ cycles. In other words, a DCJ operation is sorting if it increases the number of cycles in $\mathtt{NG}(G)$ by 1.

Given $(u \;\; v)$ an adjacency of G that is not a double-adjacency, we denote by $DCJ(u \;\; v)$ the DCJ operation that cuts adjacencies $(\overline{u} \;\; x)$ and $(y \;\; \overline{v})$ to form adjacencies $(\overline{u} \;\; \overline{v})$ and $(y \;\; x)$, making $(u \;\; v)$ a double-adjacency.

Property 7 *Let $(u \;\; v)$ be an adjacency of G that is not a double-adjacency, $DCJ(u \;\; v)$ is a sorting DCJ operation.*

Proof 18 *$DCJ(u \;\; v)$ increases the number of cycles in $\mathtt{NG}(G)$ by 1, by creating a new cycle composed of adjacencies $(u \;\; v)$ and $(\overline{u} \;\; \overline{v})$.*■

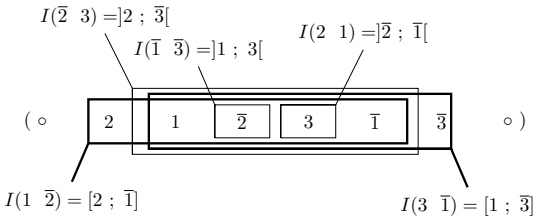


Figure 3.5: $\mathcal{I}(G) = \{]\overline{2} \;;\; \overline{1}[\,, [\mathbf{2} \;;\; \overline{\mathbf{1}}]\,, \;]2 \;;\; \overline{3}[\,, \;[\mathbf{1} \;;\; \overline{\mathbf{3}}]\,, \;]1 \;;\; 3[\}$, the set of intervals of $G = (\circ \;\; 2 \;\; 1 \;\; \overline{2} \;\; 3 \;\; \overline{1} \;\; \overline{3} \;\; \circ)$ depicted as boxes. The two boxes with thick lines represent two overlapping intervals of $\mathcal{I}(G)$ inducing a BI which exchanges 2 and $\overline{3}$.

Definition 18 *Let $(u \;\; v)$, $(\overline{u} \;\; x)$, and $(y \;\; \overline{v})$ be adjacencies of G. The* interval *of the adjacency $(u \;\; v)$, denoted by $I(u \;\; v)$ is either:*

- *the interval $[x \;;\; y]$ if $(\overline{u} \;\; x) < (y \;\; \overline{v})$. In this case, we denote it by $]\overline{u} \;;\; \overline{v}[$, or*
- *the interval $[\overline{v} \;;\; \overline{u}]$ if $(y \;\; \overline{v}) < (\overline{u} \;\; x)$.*

For example, the intervals of the adjacencies in genome $(\circ \;\; 2 \;\; 1 \;\; \overline{2} \;\; 3 \;\; \overline{1} \;\; \overline{3} \;\; \circ)$ are depicted in figure 3.5. Note that, given an adjacency $(u \;\; v)$ of G, if $(u \;\; v)$ is a double-adjacency then the interval $I(u \;\; v)$ is empty, otherwise $DCJ(u \;\; v)$ is the excision operation that extracts the interval $I(u \;\; v)$ to make it circular, thus producing the adjacency $(\overline{u} \;\; \overline{v})$.

Two intervals $I(a \; b)$ and $I(x \; y)$ are said to be *overlapping* if their intersection is non-empty, and none of the intervals is included in the other. It is easy to see, following Property 5, that given two adjacencies $(a \; b)$ and $(x \; y)$ of G such that $I(a \; b)$ and $I(x \; y)$ are non-empty intervals, the successive application of $DCJ(a \; b)$ and $DCJ(x \; y)$ is equivalent to a BI operation if and only if $I(a \;\; b)$ and $I(x \;\; y)$ are overlapping. Note that in this case neither $(a \;\; b)$, nor $(x \;\; y)$ can be double-adjacencies in G since their intervals are non-empty. Figure 3.5 shows an example of two overlapping intervals.

The following property states precisely in which case the successive application of $DCJ(a \; b)$ and $DCJ(x \; y)$ decreases the DCJ halving distance by 2, meaning that both DCJ operations are sorting.

Property 8 *Given two adjacencies $(a \;\; b)$ and $(x \;\; y)$ of G, such that $I(a \;\; b)$ and $I(x \;\; y)$ are overlapping, the successive application of $DCJ(a \;\; b)$ and $DCJ(x \;\; y)$ decreases the DCJ halving distance by 2 if and only if $x \neq \overline{a}$ and $y \neq \overline{b}$.*

Proof 19 *If $x \neq \overline{a}$ and $y \neq \overline{b}$, then the successive application of DCJ(a b) and DCJ(x y) increases the number of cycles in $\mathtt{NG}(G)$ by 2, by creating two new 2-cycles. Otherwise, DCJ(a b) first creates a new cycle that is then destroyed by DCJ(x y).*■

I denote by $\mathcal{I}(G)$, the set of intervals of all the adjacencies of G that do not contain marker $\circ$.

Remark 1 *Note that, if G contains n distinct markers, then there are $2n-1$ adjacencies in G that do not contain marker $\circ$, defining $2n-1$ intervals in $\mathcal{I}(G)$.*

Definition 19 *Two intervals $I(a\ b)$ and $I(x\ y)$ of $\mathcal{I}(G)$ are said to be* compatible *if they are overlapping and $x \neq \overline{a}$ and $y \neq \overline{b}$.*

In the following, I prove the BI 1-tandem halving distance formula by showing that if genome G contains more than three distinct markers, $n > 3$, then there exist two compatible intervals in $\mathcal{I}(G)$, and if $n = 2$ or $n = 3$ then $d^t_{BI}(G) = 1$ and $2 \leq d^p_{DCJ}(G) \leq 3$. This means that there exists a BI halving scenario S such that all BI operations in S, possibly excluding the last one, are equivalent to two successive sorting DCJ operations.

From now on, until the end of the section, $(a\ b)$ is an adjacency of G that is not a double-adjacency, A is a genome consisting in a linear chromosome $\mathfrak{L}$ and a circular chromosome $\mathfrak{C}$, obtained by applying the *sorting DCJ*, DCJ(a b), on G.

If there exists an interval $I(x\ y)$ in $\mathcal{I}(G)$ compatible with $I(a\ b)$, then applying DCJ(x y) on A consists in the integration of the circular chromosome $\mathfrak{C}$ into the linear chromosome $\mathfrak{L}$ such that the adjacency $(\overline{x}\ \overline{y})$ is formed. Such an *integration* can only be performed by cutting an adjacency $(\overline{x}\ u)$ in $\mathfrak{C}$ and an adjacency $(v\ \overline{y})$ in $\mathfrak{L}$ (or inversely) to produce adjacencies $(\overline{x}\ \overline{y})$ and $(v\ u)$. This means that there must be an adjacency $(x\ y)$ in either $\mathfrak{C}$ or $\mathfrak{L}$ such that $\overline{x}$ is in $\mathfrak{C}$ and $\overline{y}$ in $\mathfrak{L}$ or inversely. Hence, we have the following property:

Property 9 $\mathfrak{C}$ cannot *be reintegrated into $\mathfrak{L}$ by applying a sorting DCJ, DCJ(x y), on A if and only if either:*

(1) for any adjacency $(x\ y)$ in $\mathfrak{C}$ (resp. $\mathfrak{L}$), markers $\overline{x}$ and $\overline{y}$ are in $\mathfrak{L}$ (resp. $\mathfrak{C}$), or

(2) for any adjacency $(x\ y)$ in $\mathfrak{C}$ (resp. $\mathfrak{L}$), markers $\overline{x}$ and $\overline{y}$ are also in $\mathfrak{C}$ (resp. $\mathfrak{L}$).

Proof 20 *If there exists no adjacency $(x\ y)$ in A such that $\overline{x}$ is in $\mathfrak{C}$ and $\overline{y}$ in $\mathfrak{L}$ or inversely, then A necessarily satisfies either* (1)*, or* (2).■

Definition 20 *An interval $I(a\ b)$ in $\mathcal{I}(G)$ is called* interval of type 1 *(resp.* interval of type 2*) if DCJ(a b) produces a genome A satisfying configuration* (1) *(resp. configuration* (2)*) described in Property 9.*

For example, in genome $(\circ\ 2\ 1\ \overline{1}\ 3\ \overline{2}\ \overline{3}\ \circ)$, $I(\overline{1}\ 3)$ is of type 1 as $DCJ(\overline{1}\ 3)$ produces genome $(\circ\ 2\ 1\ \overline{3}\ \circ)\ (\overline{1}\ 3\ \overline{2})$; $I(\overline{2}\ \overline{3})$ is of type 2 as $DCJ(\overline{2}\ \overline{3})$ produces genome $(\circ\ 2\ 3\ \overline{2}\ \overline{3}\ \circ)\ (1\ \overline{1})$.

Now we give the maximum numbers of intervals of type 1 and type 2 that can be contained in genome G.

Lemma 7 *The maximum number of intervals of type 1 in $\mathcal{I}(G)$ is 2.*

Proof 21 *First, note that there cannot be two intervals I and J of $\mathcal{I}(G)$ such that $I \neq J$, and both I and J are of type 1. Now, if I is an interval of type 1, there can be at most two different adjacencies $(x\ y)$ and $(u\ v)$ such that $I(x\ y) = I(u\ v) = I$. In this case G necessarily has a chromosome of the form $(\ldots\ \overline{x}\ \overline{v}\ \ldots\ \overline{u}\ \overline{y}\ \ldots)$ or $(\ldots\ \overline{u}\ \overline{y}\ \ldots\ \overline{x}\ \overline{v}\ \ldots)$. Therefore, there are at most two intervals of type 1 in $\mathcal{I}(G)$.*■

Lemma 8 *The maximum number of intervals of type 2 in $\mathcal{I}(G)$ is n.*

Proof 22 *First, note that for two adjacencies $(x\ y)$ and $(\overline{x}\ z)$ in G that do not contain marker $\circ$, if $(x\ y)$ is of type 2 then $(\overline{x}\ z)$ cannot be of type 2. Now, there is only one marker u such that $(\overline{u}\ \circ)$ is an adjacency of G. Let $(u\ v)$ be the adjacency of G having u as first marker, then at most half of the intervals in $\mathcal{I}(G) - \{I(u\ v)\}$ can be of type 2. Therefore, there are at most n intervals of type 2 in $\mathcal{I}(G)$.*■

Theorem 6 *If $\mathtt{NG}(G)$ contains C cycles, then the BI 1-tandem halving distance of G is given by:*

$$d^t_{BI}(G) = \left\lfloor \frac{n - C}{2} \right\rfloor$$

Proof 23 *Since there are $2n - 1$ intervals in $\mathcal{I}(G)$, and at most $n + 2$ are of type 1 or 2, then if G contains more than three distinct markers we have $n > 3$, and since $2n - 1 > n + 2$ then there exist two compatible intervals in $\mathcal{I}(G)$ inducing a BI operation that decreases the DCJ distance by 2.*

Next, I show that if $n = 2$ or $n = 3$, then $d^t_{BI}(G) = 1$ and $2 \leq d^p_{DCJ}(G) \leq 3$.

If $n = 2$, then the genome can be written, either as $(\circ\ a\ b\ \overline{b}\ \overline{a}\ \circ)$, in which case a BI can swap a and b to produce a 1-tandem duplicated genome, or as $(\circ\ a\ \overline{a}\ b\ \overline{b}\ \circ)$, in which case a BI can swap a and $\overline{a}\ b$ to produce a 1-tandem duplicated genome.

If $n = 3$, then the genome has two double-adjacencies to be constructed, of the form $(\overline{a}\ \overline{b})$, $(\overline{x}\ \overline{y})$, with $(a\ b)$ and $(x\ y)$ being two adjacencies already present in the genome such that $b = x$ or $b = \overline{x}$ and a and y are distinct markers. One can rewrite $(a\ b)$ and $(x\ y)$ as single markers since they will not be splitted, which makes a genome with 4 markers such that at most 2 are misplaced. Then, a single BI can produce a 1-tandem duplicated genome.

Now, it is easy to see to see that if $n = 2$ or $n = 3$, then $d^p_{DCJ}(G) = n - C \leq 3$. Finally, if $n = 2$ or $n = 3$, then $d^p_{DCJ}(G) \geq 2$, otherwise we would have $d^p_{DCJ}(G) = 1$ which would imply, as G consists in a single linear chromosome, $d^t_{BI}(G) = 0$. In conclusion, if $n > 3$ then there exist two compatible intervals in $\mathcal{I}(G)$, otherwise if $n = 2$ or $n = 3$, then $d^t_{BI}(G) = 1$ and $2 \leq d^p_{DCJ}(G) \leq 3$. Therefore $d^t_{BI} = \lfloor \frac{d^p_{DCJ}}{2} \rfloor = \lfloor \frac{n-C}{2} \rfloor$.■

3.4.4.3 Sorting algorithm

In Section 3.4.4.2, I showed that if a genome G contains more than three distinct markers after reduction then there exist two compatible intervals in $\mathcal{I}(G)$ inducing a BI to perform. If G contains two or three distinct markers then the BI to perform can be trivially computed. Thus

the main concern of this section is to describe an efficient algorithm for finding compatible intervals when $n > 3$.

As in Section 3.4.4.2, in the following, G denotes a genome consisting of n distinct markers after reduction. It is easy to show that the set of intervals $\mathcal{I}(G)$ can be built in $O(n)$ time and space complexity.

We now show that finding 2 compatible intervals in $\mathcal{I}(G)$ can be done in $O(n)$ time and space complexity.

Property 10 *If $n > 3$, then all the smallest intervals in $\mathcal{I}(G)$ that are not of type 2 admit compatible intervals.*

Proof 24 *Let J be a smallest interval that is not of type 2 in $\mathcal{I}(G)$. As J is not of type 2, then J has compatible intervals if J is not of type 1.*

Let us suppose that J is of type 1, then for any adjacency $(a\ b)$ such that markers a and b are not in J, $\overline{a}$ and $\overline{b}$ are in J, and then $I(a\ b)$ is strictly included in J and $I(a\ b)$ can't be of type 2. Such adjacency does exist as there are $n > 3$ markers not included in J. Therefore J is not the smallest, which is a contradiction.■

We are now ready to give the algorithm for sorting a duplicated genome G into a 1-tandem duplicated genome with $\lfloor \frac{n-C}{2} \rfloor$ BI operations.

Algorithm 3 Reconstruction of a 1-tandem duplicated genome

```
1: while G contains more than 3 markers do
2:     Construct I(G)
3:     Pick a smallest interval I(a  b) that is not of type 2 in I(G)
4:     Find an interval I(x  y) in I(G) compatible with I(a  b)
5:     Perform the BI  equivalent to DCJ(a  b) followed by DCJ(x  y)
6:     Reduce G
7: end while
8: if G contains 2 or 3 markers then
9:     Find the last BI operation and perform it
10: end if
```

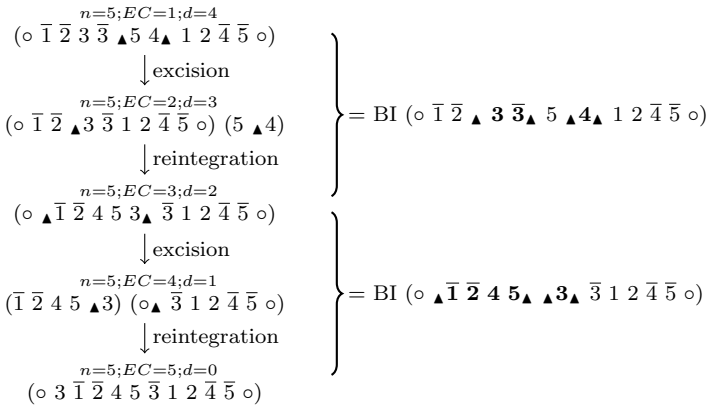
Theorem 7 *Algorithm 3 reconstructs a 1-tandem duplicated genome with a BI scenario of length $\lfloor \frac{n-C}{2} \rfloor$ in $O(n^2)$ time and space complexity, by computing pairs of sorting DCJ operations.*

Proof 25 *Building $\mathcal{I}(G)$ and finding two compatible intervals can be done in $O(n)$ time and space complexity. It follows that the while loop in the algorithm can be computed in $O(n^2)$ time and space complexity.*

Finding and performing the last BI operation when $2 \leq n \leq 3$ can be done in constant time and space complexity.

Moreover, all BI operations, possibly excluding the last one, are computed as pairs of compatible intervals, ie. pairs of sorting DCJ operations, which ensures that the length of the scenario is $\lfloor \frac{n-C}{2} \rfloor$. ■

$d_{DCJ} = n - EC - \left\lfloor \frac{OP}{2} \right\rfloor = 4$



$d_{BI} = \left\lfloor \frac{d_{DCJ}}{2} \right\rfloor = 2$

Figure 3.6: A BI scenario computed by algorithm 3.

Corollary 5 *Any BI scenario computed by Algorithm 3 is also a most parsimonious DCJ scenario, twice as long since a BI is equivalent to 2 DCJ.*

An example of scenario is shown in figure 3.6.

3.4.4.4 Conclusion

By expressing BI operations through the DCJ model I could show that restricting the scope of allowed DCJ operations under the constraint of performing only BI doesn't affect the 1-tandem halving distance.

This also means BI scenarios computed by the algorithm are in fact optimal DCJ scenarios and that it solves the DCJ 1-tandem halving problem for a subclass of genomes, namely genomes for which all markers must have the same orientation (sign) as their paralogs.

We will now see how to solve this problem for DCJ in the general case.

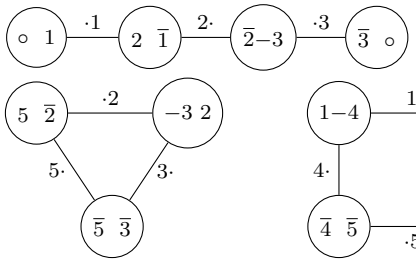
3.4.5 DCJ

In this section, because I only consider DCJ operations and no BI this time, I will drop the subscript.

$d^p(G)$ denotes the DCJ genome halving distance, towards a perfectly duplicated genome.

$d^t(G)$ denotes the DCJ 1-tandem halving distance, towards a 1-tandem duplicated genome.

I recall that $d^p(G)$ can be computed using a data structure called the *natural graph*, first introduced in [El-Mabrouk and Sankoff, 2003]. $\mathtt{NG}(G)$ is the graph whose vertices are the adjacencies of G, and 2 vertices are connected by an edge iff they share a paralogous *extremity* (see figure 3.7).



$G = (\circ\ 1\ {-4}\ \ 5\ \ \overline{2}\ {-3}\ \ 2\ \ \overline{1}\ \ \overline{4}\ \ \overline{5}\ \ \overline{3}\ \circ)$

$|OP| = 1 \qquad |EP| = 0$

$|OC| = 1 \qquad |EC| = 1$

Figure 3.7: The natural graph of G and the number of odd and even paths and cycles.

As an adjacency concerns a maximum of 2 markers extremities, this graph has a maximum degree of 2. Thus, it is composed of paths and cycles only. Moreover, it consists of nothing but 2-cycles and 1-paths if and only if G is a perfectly duplicated genome (a k-cycle or k-path is a cycle or path containing k edges). Using this graph, Mixtacki gave the following distance formula:

Theorem 8 ([Mixtacki, 2008]) *Let G be a totally duplicated genome whose natural graph contains $|\mathrm{EC}|$ even cycles and $|\mathrm{OP}|$ odd paths. Then $d^p(G) = n - |\mathrm{EC}| - \left\lfloor \frac{|\mathrm{OP}|}{2} \right\rfloor$.*

Unlike the genome halving problem, the aim of the 1-tandem halving problem is to find a 1-tandem duplicated genome. This induces one double-adjacency not to be reconstructed, which is inelegant to deal with. We will conveniently get rid of this concern.

From property 4, a 1-tandem genome that has been circularized is a perfectly duplicated genome and conversely. This allows us to establish a property that will reduce the 1-tandem halving problem to a constraint on genome halving.

Lemma 9 *Let G be a unilinear genome. Let G_c be the unicircular genome obtained by circularizing G. Then for any scenario that transforms G into a 1-tandem duplicated genome, there exists an equivalent scenario (of same length) transforming G_c into a unicircular perfectly duplicated genome, and vice versa.*

Proof 26 *As G and G_c present the same breakpoints, the scenario conversion is straightforward. It suffices to apply the same DCJ on the same breakpoints.*■

Thus, in the rest of this section, the focus will be on reconstructing an optimal perfectly duplicated genome such that it is unichromosomal. This is essentially a shape constraint on the genome halving solutions.

I will follow an approach a bit similar[9] to what has been done in [Kováč et al., 2011], as they enforced another shape constraint on optimal perfectly duplicated genome configurations. It consists in taking any optimal configuration then applying a number of successive transformations (which I will call *shapeshifting*) on it, such that they preserve the distance, and that the optimal configuration converges towards the desired shape.

In the following sections G will denote a totally duplicated genome, and G_c its circularized version. H will be an optimal perfectly duplicated genome for G_c.

[9] I had to expand on the ideas and add my own to make it a more complete system, due to the nature of the single tandem halving problem.

Following theorem 8, one can observe that circularization can alter the halving distance, depending on whether the path of $\texttt{NG}(G)$ is even or odd.

Property 11 *If G is a genome such that $\texttt{NG}(G)$ contains an even path, $d^p(G_c) = d^p(G) - 1$. Else, $d^p(G_c) = d^p(G)$.*

From Mixtacki's formula (Theorem 8), we know that optimal halving scenarios on circular genomes are scenarios which increase the number of even cycles at each step. There are two ways of increasing it. Either by splitting a cycle (*i.e.* extracting an even cycle from any cycle), or by merging two odd cycles.

As it can be quite complex at first sight, the shapeshifting system will first be described on a restricted class of genomes, namely those whose natural graph contains only even cycles. This way, we ensure that optimal halving scenarios consist only in cycle extractions. This special system will then be easily generalized to all genomes by considering cycle-merging operations.

3.4.5.1 Special shapeshifting

Here we consider that $\texttt{NG}(G_c)$ has only even cycles. It follows that $\texttt{NG}(G)$ has an even path and $d^p(G_c) = d^p(G) - 1$.

Anatomy of a multicircular perfectly duplicated genome. H is an optimal perfectly duplicated genome for G_c. Since G_c is unicircular, $\texttt{NG}(G_c)$ contains nothing but cycles. Therefore, H consists of circular chromosomes only. For H to be a perfectly duplicated genome, circular chromosomes can be of two kinds: doubled chromosomes, which can be reduced to $(x\ \overline{x})$, and single chromosomes, which can be reduced to (x) and have a *paralog chromosome* in H, which can be reduced to $(\overline{x})$. Thus the number of single chromosomes is even.

Shapeshifting. Any optimal perfectly duplicated genome H induces a class $\mathcal{C}_H$ of optimal halving scenarios (the class of all optimal DCJ scenarios transforming G_c into H). By observing the structure of G_c and H, we will look for small changes to apply to $\mathcal{C}_H$, along two criteria: H must converge toward the desired shape, and it must preserve its optimality. Such small changes are called *shapeshifters*.

In our case, we want to end up with the least number of chromosomes in H (ideally only one), therefore we will look for ways to merge chromosomes while preserving optimality. This leads us to the following definition:

Definition 21 *A shapeshifter is an adjacency $(x\ y)$ such that x and y belong to* different chromosomes *of H (convergence toward the desired shape), and such that $(x\ y)$ (and therefore $(\overline{x}\ \overline{y})$ as well) can be reconstructed by an optimal halving scenario (preservation of optimality).*

For example, if H contains markers x and y in different chromosomes, C_x and C_y, and if $(x\ y)$ can be reconstructed by an optimal halving scenario, then such scenario induces a new shape for H such that C_x and C_y cannot be distinct chromosomes anymore.

As for now we consider genomes whose natural graph has even cycles only, shapeshifters are adjacencies reconstructible by extracting even cycles.

Property 12 *Adjacencies* $(x\ y)$ *reconstructible by extracting even cycles are those such that there exists, in* $\mathtt{NG}(G_c)$*, an induced subgraph which is an* even *path, whose endpoints have outgoing edges* $x\cdot$ *and* $\cdot y$*.*

Indeed, a DCJ reconstructing $(x\ y)$ will cut at the endpoints of such path and transform it into an even cycle. However, it is not necessary to consider all even paths, so w.l.o.g we shall focus only on 2-paths (ie. adjancencies $(x\ y)$ that are *present in* G_c), which correspond to 2-cycles extractions.

For example, $(1\ 4)$ in fig. 3.7 is a shapeshifter, as the 2-path induced by vertices $(1\ {-4})$, $(\overline{1}\ \overline{4})$, and $({-4}\ 5)$ meets the requirements.

We may proceed and show how to simply apply a shapeshifter on $\mathcal{C}_H$: Let $(x\ p)$ be the adjacency containing the extremity $x\cdot$ in H, and $(q\ y)$ the one containing the extremity $\cdot y$, it suffices to perform on H one DCJ cutting adjacencies $(x\ p)$ and $(q\ y)$ to reconstruct $(x\ y)$ (and $(p\ q)$), and the equivalent DCJ on the paralogs, cutting adjacencies $(\overline{x}\ \overline{p})$ and $(\overline{q}\ \overline{y})$ to reconstruct $(\overline{x}\ \overline{y})$ (and $(\overline{p}\ \overline{q})$).

One can easily verify that the resulting genome is still optimal (first DCJ brings H closer to G_c, second one reconstructs a perfectly duplicated genome).

Now we may proceed and study the shapeshifting induced by these DCJ.

Let $(x\ y)$ be a shapeshifter in G_c. x and y belong to different chromosomes in H, so there are only 3 possible cases depending on the types of chromosomes (C_S for single chromosomes, and C_D for doubled ones) which contain these markers: 1) $x \in C_S, y \in C_D$, 2) $x, y \in C_D$, 3) $x, y \in C_S$. The last one could lead to different shapes. Figure 3.8 illustrates how the genome shape can be altered, for each case.

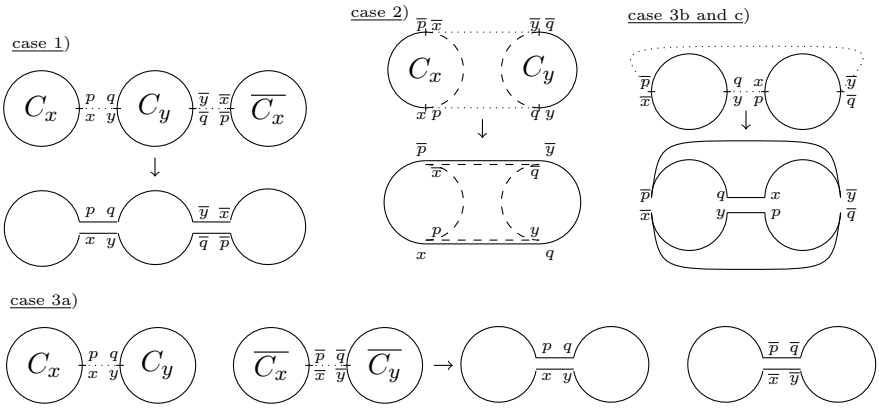


Figure 3.8: The different shapes that can be obtained by applying a shapeshifter.

More formally, one can represent shapeshifting as a system of rewriting rules:

1)	$2 \times C_S + C_D \to C_D$	3.a)	$4 \times C_S \to 2 \times C_S$	3.c)	$2 \times C_S \to 2 \times C_S$
2)	$2 \times C_D \to 2 \times C_S$	3.b)	$2 \times C_S \to 2 \times C_D$		

This is convenient as one can deduce useful properties by looking at these rules, which we are about to do, in order to study limit states of the system.

Property 13 *Shapeshifting cannot increase the number of chromosomes.*

Thus, any limit-cycle necessarily uses rules that do not change the number of chromosomes. Moreover, using rule 2 would eventually lead to using rule 3.b or 3.c as doubled chromosomes are changed into single chromosomes.

Property 14 *Any limit-cycle of the system necessarily uses rule 3.b or 3.c.*

Property 15 *Parity of $|C_D|$ is invariant by shapeshifting.*

Property 16 *A unicircular genome (ie. one doubled chromosome) is the only steady state of the system.*

Lemma 10 *By shapeshifting, the number of chromosomes in H can always be decreased under 3.*

Proof 27 *Having 3 chromosomes or more guarantees existence of shapeshifters decreasing their number. Consider the case where H contains only 2 single chromosomes C_S and $\overline{C_S}$. Label the markers from G by the chromosome which holds them in H. Adding new chromosomes necessarily creates shapeshifters between at least one of the new chromosomes and C_S or $\overline{C_S}$. Such shapeshifter decreases the number of chromosomes.*■

Lemma 11 *There exists a unicircular optimal perfectly duplicated genome for G_c if and only if H has an* odd *number of doubled chromosomes.*

Proof 28 *Straightforward from lemma 10 and property 15.*■

Lemma 12 *If H has an* even *number of doubled chromosomes, the minimum number of DCJ operations required to reconstruct a unicircular perfectly duplicated genome is $d^p(G_c)+1$, and it can always be attained.*

Proof 29 *From lemma 11, it is impossible to attain a unicircular genome in $d^p(G_c)$ operations. However, from lemma 10 and property 14, it is then always possible to attain two single chromosomes. Two single chromosomes can then be transformed into one doubled chromosome by one DCJ.*■

In conclusion, special shapeshifting allows to compute the tandem distance of any genome G such that $\texttt{NG}(G)$ contains only even cycles.

Theorem 9 *Let G be a totally duplicated genome such that $\texttt{NG}(G)$ contains only even cycles. Let G_c be its circularized version, and H any optimal perfectly duplicated genome for G_c. $d^t(G) = d^p(G)-1$ if and only if H contains an odd number of doubled chromosomes. Else $d^t(G) = d^p(G)$.*

Proof 30 *Since $\texttt{NG}(G)$ contains only even cycles, it contains an even path. Therefore from property 11, $d^p(G_c) = d^p(G) - 1$. From lemma 9 we have that $d^t(G) = d^p(G_c)$ if and only if there exists a unicircular optimal perfectly duplicated genome. Theorem then follows from lemmas 11 and 12.*■

The next step is to generalize the shapeshifting system in order to take all possible genomes into account.

3.4.5.2 General shapeshifting

As usual, G is a totally duplicated genome, G_c its circularized version, and H an optimal perfectly duplicated genome for G_c. I will also keep the same notations related to shapeshifters as in the previous section: $(x\ y)$ is a shapeshifter such that x (resp. y) is present in chromosome C_x (resp. C_y) of H, through adjacency $(x\ p)$ (resp. $(q\ y)$).

The difference with special shapeshifting is that, *in addition* to everything covered by special shapeshifting, optimal halving scenarios may now also contain cycle merges. Therefore I have to consider shapeshifters that are adjacencies which can be optimally reconstructed through merges.

Property 17 *Adjacencies $(x\ y)$ reconstructible by merges are those such that extremities $x\cdot$ and $\cdot y$ are in* two distinct odd cycles *of $\mathtt{NG}(G_c)$.*

Corresponding shapeshifters can still allow the same shapeshifting rules depending on the types of C_x and C_y. Additionally, it is now possible to have $p = \overline{y}$ and $q = \overline{x}$. This implies that $C_y = \overline{C_x}$ and induces yet another degenerated case. The general shapeshifting set of rule becomes:

1)	$2 \times C_S + C_D \to C_D$	3.a)	$4 \times C_S \to 2 \times C_S$	3.c)	$2 \times C_S \to 2 \times C_S$
2)	$2 \times C_D \to 2 \times C_S$	3.b)	$2 \times C_S \to 2 \times C_D$	**3.d)**	$\mathbf{2 \times C_S \to C_D}$

This new rule gives general shapeshifting a very interesting property.

Property 18 *Rule 3.d changes parity of C_D.*

Lemma 13 *If $\mathtt{NG}(G_c)$ contains odd cycles, and if H is made of two single chromosomes, then rule 3.d can be applied.*

Proof 31 *As $\mathtt{NG}(G_c)$ contains odd cycles, there are merges in any optimal scenario from G_c to H. Thus, there exists an adjacency $(x\ p)$ in C_x such that the adjacencies concerning extremities $x\cdot$ and $\cdot p$ are in two distinct odd cycles of $\mathtt{NG}(G_c)$. By definition, the adjacency concerning extremity $\cdot\overline{p}$ is in the same cycle as the one concerning $\cdot p$. Therefore, $(x\ \overline{p})$ is a shapeshifter inducing rule 3.d.*■

Corollary 6 *Presence of odd cycles in $\mathtt{NG}(G_c)$ ensures a unicircular optimal perfectly duplicated genome that can always be reached, as rule 3.d can always adjust the parity of C_D if needed.*

Theorem 10 *Let G be a totally duplicated genome such that $\mathtt{NG}(G)$ contains at least one odd cycle, and G_c its circularized version. Then $d^t(G) = d^p(G_c)$.*

Proof 32 *From lemma 9 we have $d^t(G) = d^p(G_c)$ iff there exists a unicircular optimal perfectly duplicated genome. Corollary from lemma 13 ensures that there does.*■

3.4.5.3 Distance

I may finally state a definite formula for the 1-tandem halving distance, as well as results on computational complexity of this problem, by gathering results from the previous sections.

Theorem 11 $d^t(G) = n - |\mathrm{EC}| - |\mathrm{EP}| + f_G$

Where f_G is a parameter that is equal to 1 iff $|C_D|$ is even and $|\mathrm{OC}| = 0$, and is equal to 0 otherwise. $|\mathrm{EC}|$, $|\mathrm{EP}|$ and $|\mathrm{OC}|$ are respectively the number of even cycles, even paths and odd cycles in $\mathtt{NG}(G)$.

Proof 33 *Straightforward from theorems 9 and 10.*■

Theorem 12 *$d^t(G)$ can be computed in linear time.*

Proof 34 *$\mathtt{NG}(G)$ can be computed in linear time, as well as an optimal perfectly duplicated genome.*■

Theorem 13 *Computing a scenario can be done in quadratic time.*

Proof 35 *An optimal perfectly duplicated genome can be computed in $O(n)$ time using Mixtacki's algorithm ([Mixtacki, 2008]). From lemma 10, one can reduce H to the minimum number of chromosomes using $O(n)$ shapeshifters. Each shapeshifter can be found in $O(n)$ time, so we have a $O(n^2)$ time shapeshifting algorithm. An optimal DCJ scenario between G and H can then be computed in $O(n)$ time using Yancopoulos' algorithm ([Yancopoulos et al., 2005]). Thus the algorithm takes quadratic time on the whole.*■

3.4.6 Closing words and credits

I was asked by Jean-Stéphane Varré to work on tandem duplications for my master's degree in 2010. I developed the single tandem halving problem as a starting point.

On halving by block interchange

Aida Ouangraoua taught me Anne Bergeron's method for sorting by DCJ as in [Bergeron et al., 2006].

Aida O. also heavily insisted I use her alternate[10] proof for the distance formula. She then rephrased one of my intermediate proofs (proof 22).

Jean-Stéphane V. drew some of the figures for the article.

I developed my BI 1-tandem halving study mainly with inspiration from [Mixtacki, 2008]. I will add I have been very admirative of Julia Mixtacki's work, in this paper and in the others, for it always presented results with very elegant proofs and reasonings. Her style has been and remains a major inspiration for me.

[10] non-working

On halving by DCJ

Jean-Stéphane V. drew some of the figures.

I developed shapeshifting with inspiration from [Kováč et al., 2011]. I would like to thank the authors of that paper as it allowed me to gain a much better insight into the space of genome halving scenarios.

Aida O. did not contribute as she was in Canada during most of the time I have been working on this. She participated in some of the early discussions, before shapeshifting was developed.

Once again I took care of proving every single result from these papers.

3.5 Model III: Partial tandem duplication

This work has been published in [Thomas et al., 2012b].

Along with the single tandem halving by DCJ, I studied other tandem models:

I studied a model where only a subset of the markers are duplicated. I could not settle complexity of this problem but provided a heuristic algorithm.

I also designed and studied various extended models were multiple tandem duplications occurred such that markers could be present in more than 2 copies.

I proved NP-hardness of all these variants.

3.5.1 Model

3.5.1.1 Considered genomes

I use duplicated genomes, perfectly duplicated genomes as defined in section 2.4, and dedoubled genomes as defined in section 3.4.2.1.

I also introduce a generalization of tandem duplicated genomes, namely *k-tandem duplicated genomes*.

Definition 22 *A* k-tandem duplicated genome *is a totally duplicated genome which can be reduced to a unilinear dedoubled genome consisting of k distinct markers.*

For example, the genome $(\circ\ 1 \diamond 2 \diamond 3\ \overline{1} \diamond \overline{2} \diamond \overline{3}\ 4 \diamond 5\ \overline{4} \diamond \overline{5}\ \circ)$ is a 2-tandem duplicated genome that can be reduced to the dedoubled genome $(\circ\ 6\ \overline{6}\ 7\ \overline{7}\ \circ)$.

Naturally, this new definition is consistent with the previous definition of a 1-tandem duplicated genome.

3.5.1.2 Considered operations

The considered operation model is the DCJ model.

3.5.2 Disrupted Single Tandem Halving

As we could solve the 1-tandem halving problem, a first direction for generalization will be considering genomes containing both duplicated and non-duplicated markers, as it is in better accordance with real biological data.

This can be seen as a 1-tandem halving problem in which adjacencies between duplicated markers can be broken by presence of non-duplicated ones. In other words, non-duplicated markers *disrupt* the 1-tandem halving.

Definition 23 *The* disrupted 1-tandem halving problem *is a variant of the 1-tandem halving problem in which the genome contains both duplicated and non-duplicated markers. The duplicated markers have to be regrouped and arranged in tandem. The corresponding distance, the* disrupted 1-tandem halving distance*, is denoted* $d^{t'}(G)$.

3.5.3 DCJ

Although a polynomial solution could not be found, in this section I describe a polynomial approximate algorithm and precise its bounds.

I suspect this problem to be NP-hard due to its relation to k-tandem halving.

3.5.3.1 Preliminary analysis.

Any optimal disrupted 1-tandem halving scenario performs two tasks: it gathers duplicated markers together (gathering phase), and it reorganizes them in a tandem (tandem phase).

Definition 24 *A* break *is an interval of non-duplicated markers surrounded by duplicated markers.*

From now on, G is a duplicated genome containing n duplicated markers separated by b breaks.

Definition 25 *A* gathering operation *is a DCJ which reduces the number of breaks in* G.

Note that the presence of excisions in the gathering phase may produce a genome consisting of multiple chromosomes. Excisions and their resulting chromosomes will be categorized depending on whether said chromosomes can be reintegrated at best in their source chromosome while increasing the number of even cycles (*good* excision/chromosome), leaving it unchanged (*neutral*) or decreasing it (*bad*). As this variation in |EC| changes the tandem distance, we get the following property.

Property 19 *Once the gathering phase is over in* G*, the remaining distance is* $d^t(G)+C^0+2C^-$*, with* C^0 *the number of neutral chromosomes and* C^- *the number of bad ones.*

The key to build an optimal disrupted 1-tandem halving scenario is to find a gathering scenario that maximizes the number of even cycles while minimizing the number of neutral and bad excisions.

3.5.3.2 Optimizing the gathering scenario.

A DCJ can decrease the number of breaks by at most 1.

Property 20 *The minimum number of gathering operations is b.*

Gathering operations are DCJ whose breakpoints are on path endpoints from $\mathtt{NG}(G)$. Breakpoints in two distinct paths will merge them, while breakpoints on the endpoints of a same path will circularize it.

Property 21 *An optimal gathering operation is one that either merges two odd paths, or circularizes an even path.*

I now give the maximum number of even cycles a set of b gathering operations can create.

Lemma 14 *A shortest gathering scenario can create up to* $\left\lfloor \frac{|\mathrm{OP}|}{2} \right\rfloor + |\mathrm{EP}| - 1$ *even cycles.*

Proof 36 *sketch of proof: Any even path can be circularized by one DCJ, while any two odd paths can be turned into two even cycles with 2 DCJs. Since b breaks induce b+1 paths in* $\mathtt{NG}(G)$*, the number of gathering operations we can use is* $b = |\mathrm{OP}| + |\mathrm{EP}| - 1$.■

Corollary 7 $d^{t'}(G) \geq n - |\mathrm{EC}| - 1 + \left\lceil \frac{|\mathrm{OP}|}{2} \right\rceil$.

This is assuming a shortest gathering phase produced no bad nor neutral chromosome, and that we are in the best case for the remaining tandem distance ($d^t(G) = d^p(G) - 1$).

Neutral excisions induce a penalty which is the same as performing a non-optimal gathering reversal, bad excisions are even worse. Thus the greedy heuristic will proceed as follows: Look for an optimal gathering operation which is a reversal or a good excision. When there is none, perform a non-optimal gathering reversal.

Let $C_h(G)$ be the number of even cycles produced by the heuristic, then we obtain the following upperbound: $d^{t'}(G) \leq n - |\mathrm{EC}| + |\mathrm{OP}| + |\mathrm{EP}| - 1 - C_h(G)$.

In the worst case, $C_h(G)$ can be equal to 0, however, the algorithm seems to perform pretty well on random genomes, giving values close to the lowerbound.

3.5.4 Beyond duplications: Multiple tandem halving

Unlike 1-tandem halving, k-tandem halving can be defined in various ways (is the content of each tandem fixed or only the number? Is the order constrained? etc...). I explored various cases, each time describing a more constrained model:

- Fixing the number of tandem to be reconstructed (k), the problem is NP-hard.
- Fixing the markers to be contained in each of the k tandem, the problem remains NP-hard.
- Fixing the order in which the tandem appear in the ancestral genome, still NP-hard.

- Lastly, a *signed* version where the relative orientation of the tandems is fixed is also NP-hard.

It is unfortunate, as multiple tandems are more relevant from a biological point of view.

Detailed study and proofs follow.

3.5.4.1 Genome Dedoubling

As k-tandem duplicated genomes can be reduced to dedoubled genomes, I will restate the *genome dedoubling problem* (already studied in section 3.3.3).

Definition 26 *Given a rearranged duplicated genome G composed of a single chromosome, the* genome dedoubling problem *consists in finding a dedoubled genome H such that the distance between G and H is minimal.*

I recall the general working of an optimal genome dedoubling algorithm, using $\texttt{DA}(G)$ (refer to 3.3.3 for definition of $\texttt{DA}(G)$ as well as detailed proofs):

1. Pick a maximum number of pairwise disjoint cycles in $\texttt{DA}(G)$.
2. Split them all into 1-cycles.
3. Extract 1-cycles concerning other markers in any way until you obtain at least n disjoint 1-cycles.
4. *(unilinear variant only)* merge all remaining cycles with the path of $\texttt{DA}(G)$.

I remind the reader the genome dedoubling problem is NP-hard, since picking a maximum number of pairwise disjoint cycles in $\texttt{DA}(G)$ is NP-hard.

Naturally the unilinear variant is NP-hard as well.

I state a similar result for a small variation on this problem as it will prove useful later.

Definition 27 *A* loosely dedoubled genome *is a unilinear totally duplicated genome G such that for each marker x, either $(x\ \overline{x})$, $(-x\ \overline{x})$, $(x\ -\overline{x})$ or $(-x\ -\overline{x})$ is an adjacency of G.*

Essentially it is a unilinear dedoubled genome in which the sign of each marker is disregarded. It means that for each marker x, $\texttt{DA}(G)$ either has one 1-cycle for x and one edge for x in the path, or 2 consecutive edges for x in the path.

Definition 28 *The* loose dedoubling problem *is a variant of the genome dedoubling problem where the aim is a loosely dedoubled genome.*

Theorem 14 *The loose genome dedoubling problem is NP-hard.*

Proof 37 *The loose variant allows one to avoid having to extract 1-cycles from the path when it presents consecutive edges for a same marker. However, in order to attain the minimum number of operation, it is still required to minimize the number of cycles to be merged with the path. In other words, one still has to pick a maximum number of pairwise disjoint cycles in $\texttt{DA}(G)$.*■

We may now proceed and study k-tandem halving problems.

3.5.4.2 Fixed tandem number

Here we just aim at reconstructing k tandems, regardless of their respective marker contents.

Definition 29 *Let G be a totally duplicated genome consisting of n distinct markers, let $0 < k \leq n$ be an integer. The* k-tandem halving *problem consists in finding a k-tandem duplicated genome H such that the distance between G and H is minimal.*

Theorem 15 *The k-tandem halving problem is NP-hard.*

Proof 38 *Genome Dedoubling problem is the particular case of k-tandem halving where $k = n$.* ■

3.5.4.3 Fixed tandem content

The goal is now to reconstruct k tandems whose respective marker contents are given.

Definition 30 *Let G be a totally duplicated genome, consisting of n distinct markers, let $P = \{P_1, P_2, ..., P_k\}$ be a partition of the set of distinct markers. The* k-fixed-tandem halving *problem consists in finding a k-tandem duplicated genome H such that each tandem is made of the markers of a P_i set, and such that the distance between G and H is minimal.*

Theorem 16 *The k-fixed-tandem halving problem is NP-hard.*

Proof 39 *Genome Dedoubling problem is the particular case of k-fixed-tandem problem where P is a set of singleton sets.* ■

3.5.4.4 Fixed tandem content and fixed tandem order

I will now constrain, additionally to the tandems content, the order in which the tandems are appearing in the final configuration.

Definition 31 *Let G be a totally duplicated genome, consisting of n distinct markers, let $P = \{P_1, P_2, ..., P_k\}$ be a partition of the set of distinct markers. The* k-ordered-tandem halving *problem consists in finding a k-tandem duplicated genome H such that consecutive tandems are made of the markers of consecutives P_i sets, and such that the distance between G and H is minimal.*

This is a very strong contraint, however the problem is still NP-hard. Let's first consider the genome dedoubling variant of this problem (ie. the case where P is a set of singleton sets).

Theorem 17 *Ordered genome dedoubling problem is NP-hard.*

Proof 40 *Constraining the markers order in a dedoubled genome is a constraint on the path of $DA(G)$. Thus, the choice of pairwise disjoint cycles remains.*■

Corollary 8 *The k-ordered-tandem halving problem is NP-hard.*

3.5.4.5 Signed k-tandem halving

I will now enforce a constraint which makes genome dedoubling polynomial, and see if it can lead to a polynomial k-tandem halving problem.

Definition 32 *The* signed dedoubling problem *is a variant of the genome dedoubling problem where the sign of each doublet (ie.* $(x\ \overline{x})$ *or* $(-x\ -\overline{x})$*) is fixed.*

Lemma 15 *The signed dedoubling problem is polynomial.*

Proof 41 *There is no more possible choice of pairwise disjoint cycles. Indeed, the sign constraint enforces a particular edge (and thus a particular cycle) to be picked.*■

I will now conduct a deeper analysis of the signed k-tandem halving problem.

Genome defragmentation Similarly to the disrupted 1-tandem-halving problem, marker subsets have to be grouped during an optimal scenario. The main difference is that there are several groups to be reconstructed, disrupting each other. Thus, *defragmentation* seems to be a more appropriate term.

Definition 33 *A* fragment *is an interval of markers from a same group, surrounded by markers from others groups* or telomeres.

Definition 34 *A* defragmentation operation *is a DCJ which reduces the number of fragments in G.*

Lemma 16 *Computing the minimum number of defragmentation operations is NP-hard.*

Proof 42 *Any loose dedoubling problem instance can be seen as a defragmentation problem under the constraint that each group is split in no more than 2 fragments (one marker stands for a fragment in a genome).*■

Theorem 18 *Signed k-tandem halving problem is NP-hard.*

Proof 43 *This is proven by reduction, from the problem of computing the minimum number of defragmentation operations, to a subclass of signed k-tandem halving. Consider the class of genomes for which there exists an optimal scenario consisting only of a defragmentation phase. Theorem then follows from lemma 16.*■

3.5.5 Closing words and credits

On disrupted tandem halving

I recall I originally developed the single tandem halving problem as a starting point for disrupted tandem halving.

Jean-Stéphane V. drew some of the figures for the article.

On multiple tandem halving

I developed, studied and proved all of these variants alone.

Again, Aida O. did not participate as she was in Canada while I worked on this paper.

As usual I took care of proving every single result from this paper.

3.6 Conclusion

To conclude this section, here is a table containing all of my results.

Problem	Input genome	Goal configuration
Genome Dedoubling	Totally duplicated	Dedoubled
1-tandem Halving	Totally duplicated	1-tandem
k-tandem Halving	Totally duplicated	k tandems
disrupted 1-tandem Halving	Duplicated	Tandem

Problem	Operation model	Complexity
Genome Dedoubling	DCJ or Reversal	NP-hard (FPT in the number of cycles)
1-tandem Halving	DCJ or Block Interchange	$O(n)$ (scenario in $O(n^2)$)
k-tandem Halving	DCJ	NP-hard
disrupted 1-tandem Halving	DCJ	open

General conclusion

I think my PhD thesis could be summarized by the following sentence:

"Biological reality is NP-hard, polynomial problems come with a catch: they don't make much sense".

The catch being no duplicated content, or an exponential number of different genomes that are all optimal, or other aspects causing awkward moments in bioinformatics conferences when biologists ask questions.

From a computer science and mathematical point of view, however, it's much more interesting.

We've seen that data structures are a powerful tool to shift focus.

On this matter I'll add that any algorithm can be seen as a data structure itself, and this allows a certain freedom in splitting complexity : for example in some cases an $O(n^4)$ time algorithm could be implemented as a loop of $O(n^2)$ steps each using a structure built in $O(n^2)$, or a loop of $O(n^3)$ steps using a linear structure instead... this kind of reasoning is what allowed me to find the "magic" property ("the smallest overlapping interval is an optimal operation") for single-tandem halving by block interchange.

I'll conclude by saying that this work on the whole is meant to be seen as a starting point in the study of alternate explanative models for the presence of replicated markers. As said models are taken directly from biological studies, they are obviously not new. However, it is the first time they are studied in the context of classical rearrangement problems.

While most people would notice the diversity in mathematical proofs throughout my papers, I find it is even more stimulating to try to grasp the subtle underlying similarities they must share, since down the line they really are the expression of different analyses for similar problems.

Obvious further perspectives would be the study of other duplication hypotheses, or combination of previously studied ones, even though I feel the mastery of previously studied models and their behavior towards multiple operation models should take priority.

For example, the attempt at solving genome dedoubling by reversal with unoriented genomes (cf. section 3.3.5.4) was never published and the reason is that the answer we seemed to find (computing an optimal reversal scenario through dynamic programming, in polynomial time) was not a satisfying one. My real goal with this work was to be able to directly compute the orientation cost without the need for building an optimal scenario, just as it was done for classical sorting by reversal in [Bergeron et al., 2004]. I find this kind of result allows a much deeper understanding of the studied problem.

In the same line of thought, I'd consider that even the classical genome halving by reversal, as solved in [El-Mabrouk et al., 1998], doesn't provide a satisfying answer and would deserve further studies.

More generally, I think that while operational research provides very interesting concepts to tackle hard problems, it should be reserved, as intended, to hard problems, be it NP-hard rearrangement problems or software application meant to process very large genomes. In the context of theoretical papers, finding a polynomial answer through such techniques should be seen as an encouragement to find a more elegant method that would provide a better understanding of the problem.

Finally, to give a few words about where I see this field going in a couple years, I would say that unless major changes occur, I do not foresee a bright future in rearrangements for the LIFL, given the type of people working in bioinformatics there. Even though I wish them success in the name of scientific progress, I don't think it can be done by blatantly trashing ethics, claiming ownership of other people's work by nothing more than writing their names at the highest possible place on the paper, somewhat reminiscent of the way a dog would leave its mark on a fire hydrant.

Bibliography

[Alekseyev and Pevzner, 2007] Alekseyev, M. A. and Pevzner, P. A. (2007). Whole genome duplications, multi-break rearrangements, and genome halving problem. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 665–679. Society for Industrial and Applied Mathematics.

[Alekseyev and Pevzner, 2008] Alekseyev, M. A. and Pevzner, P. A. (2008). Multi-break rearrangements and chromosomal evolution. *Theoretical Computer Science*, 395(2):193–202.

[Angibaud et al., 2007] Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., and Vialette, S. (2007). A pseudo-boolean programming approach for computing the breakpoint distance between two genomes with duplicate genes. In *Comparative Genomics*, pages 16–29. Springer.

[Bader et al., 2001] 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. *Journal of Computational Biology*, 8:483–491.

[Bafna and Pevzner, 1998] Bafna, V. and Pevzner, P. (1998). Sorting by transpositions. *SIAM Journal on Discrete Mathematics*.

[Bafna and Pevzner, 1995] Bafna, V. and Pevzner, P. A. (1995). Sorting permutations by transpositions. In Clarkson, K. L., editor, *SODA*, pages 614–623. ACM/SIAM.

[Bailey et al., 2004] Bailey, J., Baertsch, R., Kent, W., Haussler, D., and Eichler, E. (2004). Hotspots of mammalian chromosomal evolution. *Genome Biology*, 5(4):R23.

[Bérard et al., 2007] Bérard, S., Bergeron, A., Chauve, C., and Paul, C. (2007). Perfect sorting by reversals is not always difficult. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 4(1):4–16.

[Bérard et al., 2008] Bérard, S., Chauve, C., and Paul, C. (2008). A more efficient algorithm for perfect sorting by reversals. *Information processing letters*, 106(3):90–95.

[Bérard et al., 2012] Bérard, S., Gallien, C., Boussau, B., Szöllősi, G. J., Daubin, V., and Tannier, E. (2012). Evolution of gene neighborhoods within reconciled phylogenies. *Bioinformatics*, 28(18):i382–i388.

[Bergeron, 2001] Bergeron, A. (2001). A very elementary presentation of the hannenhalli-pevzner theory. *Lecture Notes in Computer Science*, 2089:106–117.

[Bergeron, 2005] Bergeron, A. (2005). A very elementary presentation of the hannenhalli-pevzner theory. *Discrete Applied Mathematics*, 146(2):134–145.

[Bergeron et al., 2004] Bergeron, A., Mixtacki, J., and Stoye, J. (2004). Reversal distance without hurdles and fortresses. In *In proc. of Combinatorial Pattern Matching. LNCS 3109*, pages 388–399. Springer-Verlag.

[Bergeron et al., 2005] Bergeron, A., Mixtacki, J., and Stoye, J. (2005). On sorting by translocations. In Miyano, S., Mesirov, J., Kasif, S., Istrail, S., Pevzner, P., and Waterman, M., editors, *Research in Computational Molecular Biology*, volume 3500 of *Lecture Notes in Computer Science*, pages 615–629. Springer Berlin Heidelberg.

[Bergeron et al., 2006] Bergeron, A., Mixtacki, J., and Stoye, J. (2006). A unifying view of genome rearrangements. In Bücher, P. and Moret, B., editors, *Algorithms in Bioinformatics*, volume 4175 of *Lecture Notes in Computer Science*, pages 163–173. Springer Berlin Heidelberg.

[Bergeron et al., 2008] Bergeron, A., Mixtacki, J., and Stoye, J. (2008). Hp distance via double cut and join distance. In *Combinatorial Pattern Matching*, pages 56–68. Springer.

[Berman and Fujito, 1995] Berman, P. and Fujito, T. (1995). Approximating independent sets in degree 3 graphs. In *In proc. of Workshop on Algorithms and Data Structures. LNCS 955*, pages 449–460. Springer-Verlag.

[Biller et al., 2013] Biller, P., Feijão, P., and Meidanis, J. (2013). Rearrangement-based phylogeny using the single-cut-or-join operation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 10(1):122–134.

[Blin et al., 2007] Blin, G., Chauve, C., Fertin, G., Rizzi, R., and Vialette, S. (2007). Comparing genomes with duplications: a computational complexity point of view. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(4):523–534.

[Blin et al., 2004] Blin, G., Fertin, G., Chauve, C., et al. (2004). The breakpoint distance for signed sequences. In *1st Conference on Algorithms and Computational Methods for biochemical and Evolutionary Networks (CompBioNets' 04)*, volume 3, pages 3–16.

[Blin et al., 2009] Blin, G., Fertin, G., Sikora, F., and Vialette, S. (2009). The exemplar breakpoint distance for non-trivial genomes cannot be approximated. In *WALCOM: Algorithms and Computation*, pages 357–368. Springer.

[Braga et al., 2010] Braga, M., Willing, E., and Stoye, J. (2010). Genomic distance with dcj and indels. In Moulton, V. and Singh, M., editors, *Algorithms in Bioinformatics*, volume 6293 of *Lecture Notes in Computer Science*, pages 90–101. Springer Berlin Heidelberg.

[Braga, 2009] Braga, M. D. (2009). baobabluna: the solution space of sorting by reversals. *Bioinformatics*, 25(14):1833–1835.

[Braga et al., 2007] Braga, M. D., Sagot, M.-F., Scornavacca, C., and Tannier, E. (2007). The solution space of sorting by reversals. In *Bioinformatics Research and Applications*, pages 293–304. Springer.

[Braga and Stoye, 2010] Braga, M. D. and Stoye, J. (2010). The solution space of sorting by dcj. *Journal of Computational Biology*, 17(9):1145–1165.

[Braga and Stoye, 2013] Braga, M. D. and Stoye, J. (2013). Restricted dcj-indel model revisited. In *Advances in Bioinformatics and Computational Biology*, pages 36–46. Springer.

[Bryant, 1998] Bryant, D. (1998). The complexity of the breakpoint median problem. *Centre de recherches mathematiques*.

[Bryant, 2000] Bryant, D. (2000). The complexity of calculating exemplar distances. In *Comparative Genomics*, pages 207–211. Springer.

[Bulteau et al., 2010] Bulteau, L., Fertin, G., and Rusu, I. (2010). Sorting by transpositions is difficult.

[Bulteau et al., 2012] Bulteau, L., Fertin, G., and Rusu, I. (2012). Pancake flipping is hard. In *MFCS*, pages 247–258.

[Caprara, 1997] Caprara, A. (1997). Sorting by reversals is difficult. In *Proceedings of the First Annual International Conference on Computational Molecular Biology*, RECOMB '97, pages 75–83, New York, NY, USA. ACM.

[Caprara, 2003] Caprara, A. (2003). The reversal median problem. *INFORMS Journal on Computing*, 15(1):93–113.

[Chen, 2010] Chen, X. (2010). On sorting permutations by double-cut-and-joins. In Thai, M. and Sahni, S., editors, *Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 439–448. Springer Berlin Heidelberg.

[Chen et al., 2005] Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., and Jiang, T. (2005). Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 2(4):302–315.

[Christie, 1996] Christie, D. A. (1996). Sorting permutations by block-interchanges. *Inf. Process. Lett.*, 60(4):165–169.

[Dobzhansky and Sturtevant, 1938] Dobzhansky, T. and Sturtevant, A. H. (1938). Inversions in the Chromosomes of Drosophila Pseudoobscura. *Genetics*, 23(1):28–64.

[Dweighter, 1975] Dweighter, H. (1975). American mathematical monthly 82.

[El-Mabrouk et al., 1998] El-Mabrouk, N., Nadeau, J. H., and Sankoff, D. (1998). Genome halving. In Farach-Colton, M., editor, *Proceedings of CPM'98*, volume 1448 of *Lecture Notes in Computer Science*, pages 235–250. Springer.

[El-Mabrouk and Sankoff, 2003] El-Mabrouk, N. and Sankoff, D. (2003). The reconstruction of doubled genomes. *SIAM J. Comput.*, 32(3):754–792.

[El-Mabrouk and Sankoff, 2012] El-Mabrouk, N. and Sankoff, D. (2012). Analysis of gene order evolution beyond single-copy genes. In *Evolutionary Genomics*, pages 397–429. Springer.

[Feijão and Meidanis, 2009] Feijão, P. and Meidanis, J. (2009). Scj: a variant of breakpoint distance for which sorting, genome median and genome halving problems are easy. In *Algorithms in Bioinformatics*, pages 85–96. Springer.

[Feijao and Meidanis, 2011] Feijao, P. and Meidanis, J. (2011). Scj: a breakpoint-like distance that simplifies several rearrangement problems. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 8(5):1318–1329.

[Feng and Zhu, 2007] Feng, J. and Zhu, D. (2007). Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Transactions on Algorithms (TALG)*, 3(3):25.

[Fertin et al., 2009] Fertin, G., Labarre, A., Rusu, I., Tannier, E., and Vialette, S. (2009). *Combinatorics of genome rearrangements*. MIT press.

[Gates and Papadimitriou, 1979] Gates, W. H. and Papadimitriou, C. (1979). Bounds for sorting by prefix reversal. *Discrete Mathematics*, pages 47–57.

[Hannenhalli, 1995] Hannenhalli, S. (1995). Polynomial-time algorithm for computing translocation distance between genomes. In *Combinatorial Pattern Matching*, pages 162–176. Springer.

[Hannenhalli and Pevzner, 1995a] Hannenhalli, S. and Pevzner, P. (1995a). Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Journal of the ACM*, pages 178–189. ACM Press.

[Hannenhalli and Pevzner, 1995b] Hannenhalli, S. and Pevzner, P. A. (1995b). Transforming men into mice (polynomial algorithm for genomic distance problem. In *In 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 581–592.

[Hannenhalli and Pevzner, 1995c] Hannenhalli, S. and Pevzner, P. A. (1995c). Transforming men into mice (polynomial algorithm for genomic distance problem). In *In proc. of FOCS 1995*, pages 581–592. IEEE Press.

[Hartman and Shamir, 2004] Hartman, T. and Shamir, R. (2004). A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Information and Computation*, 204:156–169.

[Hochbaum, 2004] Hochbaum, D. S. (2004). Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254.

[Howarth et al., 2011] Howarth, K. D., Pole, J. C. M., Beavis, J. C., Batty, E. M., Newman, S., Bignell, G. R., and Edwards, P. A. W. (2011). Large duplications at reciprocal translocation breakpoints that might be the counterpart of large deletions and could arise from stalled replication bubbles. *Genome Research*, 21(4):525–534.

[Jean and Nikolski, 2007] Jean, G. and Nikolski, M. (2007). Genome rearrangements: a correct algorithm for optimal capping. *Information Processing Letters*, 104(1):14–20.

[Jiang, 2011] Jiang, M. (2011). The zero exemplar distance problem. *Journal of Computational Biology*, 18(9):1077–1086.

[Kaplan et al., 1997] Kaplan, H., Shamir, R., and Tarjan, R. E. (1997). A faster and simpler algorithm for sorting signed permutations by reversals.

[Kaplan and Verbin, 2005] Kaplan, H. and Verbin, E. (2005). Sorting signed permutations by reversals, revisited. *Journal of Computer and System Sciences*, 70(3):321–341.

[Kececioglu and Sankoff, 1993] Kececioglu, J. and Sankoff, D. (1993). Exact and approximation algorithms for the inversion distance between two chromosomes. In *Combinatorial Pattern Matching*, pages 87–105. Springer.

[Kececioglu and Ravi, 1995] Kececioglu, J. D. and Ravi, R. (1995). Of mice and men: Algorithms for evolutionary distances between genomes with translocation. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 604–613. Society for Industrial and Applied Mathematics.

[Kováč et al., 2011] Kováč, J., Warren, R., Braga, M. D., and Stoye, J. (2011). Restricted DCJ model: rearrangement problems with chromosome reincorporation. *Journal of Computational Biology*, 18(9):1231–1241.

[Kovac, 2011] Kovac, J. (2011). On the complexity of rearrangement problems under the breakpoint distance. *arXiv preprint arXiv:1112.2172*.

[Kovác et al., 2010] Kovác, J., Braga, M. D. V., and Stoye, J. (2010). The problem of chromosome reincorporation in DCJ sorting and halving. In Tannier, E., editor, *RECOMB-CG*, volume 6398 of *Lecture Notes in Computer Science*, pages 13–24. Springer.

[Labarre and Cibulka, 2011] Labarre, A. and Cibulka, J. (2011). Polynomial-time sortable stacks of burnt pancakes. *Theor. Comput. Sci.*, 412(8-10):695–702.

[Lin et al., 2005] Lin, Y. C., Lu, C. L., Chang, H.-Y., and Tang, C. Y. (2005). An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. *Journal of Computational Biology*, 12(1):102–112.

[Matzkin et al., 2005] Matzkin, L., Merritt, T., Zhu, C.-T., and Eanes, W. (2005). The structure and population genetics of the breakpoints associated with the cosmopolitan chromosomal inversion in(3r)payne in drosophila melanogaster. *Genetics*, 170:1143–1152.

[Meisel, 2009] Meisel, R. (2009). Repeat mediated gene duplication in the drosophila pseudoobscura genome. *Gene*, 438(1-2):1–7.

[Miklos et al., 2013] Miklos, I., Kiss, S. Z., and Tannier, E. (2013). On sampling scj rearrangement scenarios. *arXiv preprint arXiv:1304.2170*.

[Mixtacki, 2008] Mixtacki, J. (2008). Genome halving under DCJ revisited. In Hu, X. and Wang, J., editors, *Proceedings of COCOON'08*, volume 5092 of *Lecture Notes in Computer Science*, pages 276–286. Springer.

[Ohlebusch et al., 2005] Ohlebusch, E., Abouelhoda, M., Hockel, K., and Stallkamp, J. (2005). The median problem for the reversal distance in circular bacterial genomes. In Apostolico, A., Crochemore, M., and Park, K., editors, *Combinatorial Pattern Matching*, volume 3537 of *Lecture Notes in Computer Science*, pages 116–127. Springer Berlin Heidelberg.

[Ozery-Flato and Shamir, 2003] Ozery-Flato, M. and Shamir, R. (2003). Two notes on genome rearrangement. *Journal of Bioinformatics and Computational Biology*, 1(01):71–94.

[Ozery-Flato and Shamir, 2006] Ozery-Flato, M. and Shamir, R. (2006). Sorting by translocations via reversals theory. In Bourque, G. and El-Mabrouk, N., editors, *Comparative Genomics*, volume 4205 of *Lecture Notes in Computer Science*, pages 87–98. Springer Berlin Heidelberg.

[Palmer and Herbon, 1988] Palmer, J. D. and Herbon, L. A. (1988). Plant mitochondrial dna evolved rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, 28(1-2):87–97.

[Pe'er and Shamir, 1998] Pe'er, I. and Shamir, R. (1998). The median problems for breakpoints are np-complete. In *Elec. Colloq. on Comput. Complexity*, volume 71.

[Ranz et al., 2007] Ranz, J., Maurin, D., Chan, Y., and Von Grotthuss, M. (2007). Principles of genome evolution in the Drosophila melanogaster species group. *PLoS biology*, 5(6):e152+.

[Richards et al., 2005] Richards, S., Liu, Y., Bettencourt, B., Hradecky, P., and Letovsky, S. (2005). Comparative genome sequencing of drosophila pseudoobscura: Chromosomal, gene, and cis-element evolution. *Genome Research*, 15:1–18.

[Sagot and Tannier, 2005] Sagot, M.-F. and Tannier, E. (2005). Perfect sorting by reversals. In *Computing and Combinatorics*, pages 42–51. Springer.

[Sankoff, 1989] Sankoff, D. (1989). Mechanisms of genome evolution: models and inference. *Bull. Int. Stat. Instit*, 47:461–475.

[Sankoff, 1999] Sankoff, D. (1999). Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917.

[Sankoff and Blanchette, 1998] Sankoff, D. and Blanchette, M. (1998). Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5(3):555–570.

[Sturtevant, 1921] Sturtevant, A. (1921). Genetic studies on drosophila simulans. iii. autosomal genes. general discussion. *Genetics*, 6(2):179.

[Sturtevant and Novitski, 1941] Sturtevant, A. and Novitski, E. (1941). The homologies of the chromosome elements in the genus drosophila. *Genetics*, 26(5):517.

[Sturtevant and Dobzhansky, 1936] Sturtevant, A. H. and Dobzhansky, T. (1936). Inversions in the Third Chromosome of Wild Races of Drosophila Pseudoobscura, and Their Use in the Study of the History of the Species. *Proceedings of the National Academy of Sciences of the United States of America*, 22(7):448–450.

[Tannier et al., 2007] Tannier, E., Bergeron, A., and Sagot, M.-F. (2007). Advances on sorting by reversals. *Discrete Applied Mathematics*, 155(6):881–888.

[Tannier and Sagot, 2004] Tannier, E. and Sagot, M.-F. (2004). Sorting by reversals in subquadratic time. In *Combinatorial pattern matching*, pages 1–13. Springer.

[Tannier et al., 2008] Tannier, E., Zheng, C., and Sankoff, D. (2008). Multichromosomal genome median and halving problems. In Crandall, K. A. and Lagergren, J., editors, *Proceedings of WABI'08*, volume 5251 of *Lecture Notes in Computer Science*, pages 1–13. Springer.

[Tannier et al., 2009] Tannier, E., Zheng, C., and Sankoff, D. (2009). Multichromosomal median and halving problems under different genomic distances. *BMC bioinformatics*, 10(1):120.

[Thomas et al., 2012a] Thomas, A., Ouangraoua, A., and Varré, J.-S. (2012a). Genome halving by block interchange. In Schier, J., Correia, C. M. B. A., Fred, A. L. N., and Gamboa, H., editors, *BIOINFORMATICS*, pages 58–65. SciTePress.

[Thomas et al., 2012b] Thomas, A., Ouangraoua, A., and Varré, J.-S. (2012b). Tandem halving problems by dcj. In *Algorithms in Bioinformatics*, pages 417–429. Springer.

[Thomas et al., 2013] Thomas, A., Ouangraoua, A., and Varré, J.-S. (2013). Single tandem halving by block interchange. In *Biomedical Engineering Systems and Technologies*, pages 162–174. Springer.

[Thomas et al., 2011] Thomas, A., Varré, J.-S., and Ouangraoua, A. (2011). Genome dedoubling by dcj and reversal. *BMC bioinformatics*, 12(Suppl 9):S20.

[Warren and Sankoff, 2008] Warren, R. and Sankoff, D. (2008). Genome halving with double cut and join. In Brazma, A., Miyano, S., and Akutsu, T., editors, *Proceedings of APBC'08*, volume 6 of *Adv. in Bioinformatics and Comp. Biol.*, pages 231–240. Imperial College Press.

[Warren and Sankoff, 2009] Warren, R. and Sankoff, D. (2009). Genome aliquoting with double cut and join. *BMC bioinformatics*, 10(Suppl 1):S2.

[Warren and Sankoff, 2011] Warren, R. and Sankoff, D. (2011). Genome aliquoting revisited. *Journal of Computational Biology*, 18(9):1065–1075.

[Watterson et al., 1982] Watterson, G., Ewens, W., Hall, T., and Morgan, A. (1982). The chromosome inversion problem.

[Yancopoulos et al., 2005] Yancopoulos, S., Attie, O., and Friedberg, R. (2005). Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346.

[Yancopoulos and Friedberg, 2008] Yancopoulos, S. and Friedberg, R. (2008). Sorting genomes with insertions, deletions and duplications by dcj. In Nelson, C. and Vialette, S., editors, *Comparative Genomics*, volume 5267 of *Lecture Notes in Computer Science*, pages 170–183. Springer Berlin Heidelberg.

[Zheng et al., 2008] Zheng, C., Zhu, Q., Adam, Z., and Sankoff, D. (2008). Guided genome halving: hardness, heuristics and the history of the hemiascomycetes. *Bioinformatics*, 24(13):i96–i104.

[Zheng et al., 2006] Zheng, C., Zhu, Q., and Sankoff, D. (2006). Genome halving with an outgroup. *Evolutionary bioinformatics online*, 2:295.

List of Figures

Résumé

La compréhension de la dynamique des réarrangements génomiques est une problématique importante en phylogénie. La phylogénie est l'étude de l'évolution des espèces. Un but majeur est d'établir les relations d'évolution au sein d'un groupe d'espèces, pour déterminer la topologie de l'arbre d'évolution formé par ce groupe et des ancêtres communs à certains sous-ensembles.

Pour ce faire, il est naturellement très utile de disposer d'un moyen d'évaluer les distances évolutionnaires relatives entre des espèces, ou encore d'être capable d'inférer à un groupe d'espèces le génome d'un ancêtre commun à celles-ci.

Ce travail de thèse, dans la lignée d'autres travaux, consiste à élaborer de tels moyens, ici dans des cas particuliers où les génomes possèdent des gènes en multiples copies, ce qui complique les choses.

Plusieurs hypothèse explicatives de la présence de duplications ont été considérées, des formules de distance ainsi que des algorithmes de calcul de scénarios ont été élaborés, accompagnés de preuves de complexité.

Mots-clés: bioinformatique, génomique comparative, réarrangements, marqueurs dupliqués, genome halving, duplication en tandem, breakpoints, inversion, DCJ, échange de blocs

---

Abstract

Understanding the dynamics of genome rearrangements is a major issue of phylogenetics. Phylogenetics is the study of species evolution. A major goal of the field is to establish evolutionary relationships within groups of species, in order to infer the topology of an evolutionary tree formed by this group and common ancestors to some of these species.

In this context, having means to evaluate relative evolutionary distances between species, or to infer common ancestor genomes to a group of species would be of great help.

This work, in the vein of other studies from the past, aims at designing such means, here in the particular case where genomes present multiple occurrencies of genes, which makes things more complex.

Several hypotheses accounting for the presence of duplications were considered. Distances formulae as well as scenario computing algorithms were established, along with their complexity proofs.

Keywords: bioinformatics, comparative genomics, rearrangement, replicated markers, genome halving, tandem duplication, breakpoints, reversal, DCJ, block interchange