Algorithms for genome comparison applied to bacterial genomes

Raluca Uricaru

Académie de Montpellier
Université Montpellier II
— Sciences et Techniques du Languedoc —

Thèse

présentée au Laboratoire d'Informatique de Robotique
et de Microélectronique de Montpellier

Spécialité : **Informatique**
Formation Doctorale : **Informatique**
École Doctorale : **Information, Structures, Systèmes**

Algorithmes de comparaison de génomes appliqués aux génomes bactériens

par

Raluca Uricaru

Soutenue le 14 Décembre 2010, devant le jury composé de :

Marie-France Sagot	Rapporteur	Directeur de Recherche	INRIA Grenoble
Mathieu Blanchette	Rapporteur	Associate Professor	McGill University, Montréal
Thérèse Commes	Examinateur	Professeur des Universités	IGH Montpellier
Hélène Chiapello	Examinateur	Ingénieur de Recherche	INRA Jouy en Josas
Thomas Faraut	Examinateur	Chargé de Recherche	INRA Toulouse
Vincent Ranwez	Examinateur	Maître de Conférences	Université Montpellier II
Eric Rivals	Directeur	Directeur de Recherche	CNRS Montpellier

Abstract

With more than 1000 complete genomes available (among which, the vast majority come from bacteria), comparative genomic analysis become essential for the functional annotation of genomes, the understanding of their structure and evolution and have applications in phylogenomics or vaccine design. One of the main approaches for comparing genomes is by aligning their DNA sequences, *i.e. whole genome alignment* (WGA), which means identifying the similarity regions without any prior annotation knowledge. Despite the significant improvements during the last years, reliable tools for WGA and methodology for estimating its quality, in particular for bacterial genomes, still need to be designed. Besides their extremely large lengths that make classical dynamic programming alignment methods unsuitable, aligning whole genomes involves several additional difficulties, due to the mechanisms through which genomes evolve: the divergence, which let sequence similarity vanish over time, the reordering of genomic segments (rearrangements), or the acquisition of external genetic material generating regions that are unalignable between sequences, *e.g.* horizontal gene transfer, phages. Therefore, whole genome alignment tools implement heuristics, among which the most common is the *anchor based strategy*. It starts by detecting an initial set of similarity regions (phase 1), and, through a chaining phase (phase 2), selects a non-overlapping maximum-weighted, usually collinear, subset of those similarities, called anchors. Phases 1 and 2 are recursively applied on yet unaligned regions (phase 3). The last phase (phase 4) consists in systematically applying classical alignment tools to all short regions still left unaligned.

This thesis addresses several problems related to whole genome alignment: the evaluation of the quality of results given by WGA tools and the improvement of the classical anchor based strategy. We first designed a protocol for evaluating the quality of alignment results, based on both computational and biological measures. An evaluation of the results given by two state of the art WGA tools on pairs of intra-species bacterial genomes revealed their shortcomings: the failure of detecting some of the similarities between sequences and the misalignment of some regions. Based on these results, which imply a lack in both sensitivity and specificity, we propose a novel, pairwise whole genome alignment tool, YOC, implementing a simplified two-phase version of the anchor strategy. In phase 1, YOC improves sensitivity by using as anchors, for the first time, *local similarities* based on spaced seeds that are capable of detecting larger similarity regions in divergent sequences. This phase is followed by a chaining method adapted to local similarities, *a novel type of collinear chaining, allowing for proportional overlaps*. We give a formulation for this novel problem and provide the first algorithm for it. The algorithm, implementing a dynamic programming approach based on the sweep-line paradigm, is exact and runs in quadratic time. We show

that, compared to classical collinear chaining, chaining with overlaps improves on real bacterial data, while remaining almost as efficient in practice. Our novel tool, YOC, is evaluated together with other four WGA tools on a dataset composed of 694 pairs of intra-species bacterial genomes. The results show that YOC improves on divergent cases by detecting more distant similarities and by avoiding misaligned regions. In conclusion, YOC should be easier to apply automatically and systematically to incoming genomes, for it does not require a post-filtering step to detect misalignment and is less complex to calibrate.

Keywords: comparative genomics, whole genome alignment, anchor based strategy, spaced seeds, fragment chaining, dynamic programming, trapezoid graphs

Résumé

Avec plus de 1000 génomes complets disponibles (la grande majorité venant de bactéries), les analyses comparatives de génomes deviennent indispensables pour leur annotation fonctionnelle, ainsi que pour la compréhension de leur structure et leur évolution, et s'appliquent par exemple en phylogénomique ou au design des vaccins. L'une des approches les plus utilisées pour comparer des génomes est l'alignement de leurs séquences d'ADN, *i.e. alignement de génomes complets*, c'est-à-dire identifier les régions de similarité en s'affranchissant de toute annotation. Malgré des améliorations significatives durant les dernières années, des outils performants pour cette approche ainsi que des méthodes pour l'estimation de la qualité des résultats qu'elle produit, en particulier sur les génomes bactériens, restent encore à développer. Outre leurs grandes tailles qui rendent les solutions classiques basées sur la programmation dynamique inutilisables, l'alignement de génomes complets pose des difficultés supplémentaires dues à des mécanismes d'évolution particuliers: la divergence, qui estompe les similarités entre les séquences, le réordonnancent des portions génomiques (réarrangements), ou l'acquisition de matériel génétique extérieur, qui produit des régions non alignables entres les séquences, *e.g.* transfert horizontal des gènes, phages. En conséquence, les solutions pour l'alignement de génomes sont des heuristiques, dont la plus commune est la *stratégie basée sur des ancres*. Cette stratégie commence par identifier un ensemble initial de régions de similarité (phase 1). Ensuite une phase de chaînage sélectionne un sous-ensemble (non-chevauchantes et généralement colinéaires) de ces similarités de poids maximal, nommées ancres (phase 2). Les phases 1 et 2 sont appliquées de manière récursive sur les régions encore non-alignées (phase 3). La dernière phase consiste en l'application systématique des outils d'alignement classiques sur toutes les régions courtes qui n'ont pas encore été alignées.

Cette thèse traite plusieurs problèmes liés à l'alignement de génomes complets dont: l'évaluation de la qualité des résultats produits par les outils d'alignement et l'amélioration de la stratégie basée sur des ancres. Premièrement, nous avons créé un protocole pour évaluer la qualité des résultats d'alignement, comprenant des mesures de calcul quantitatives et qualitatives, dont certaines basées sur des connaissances biologiques. Une analyse de la qualité des alignements produits par deux des principaux outils existants sur des paires de génomes bactériens intra-espèces révèle leurs limitations: des similarités non détectées et des portions d'alignement incorrectes. À partir de ces résultats, qui suggèrent un manque de sensibilité et spécificité, nous proposons un nouvel outil pour l'alignement deux à deux de génomes complets, YOC, qui implémente une version simplifiée de la stratégie basée sur des ancres, contenant seulement deux phases. Dans la phase 1, YOC améliore la sensibilité en utilisant comme ancres, pour la première fois dans cette stratégie, des *similarités locales* basées

sur des graines espacées, capables de détecter des similarités plus longues dans des régions plus divergentes. Cette phase est suivie par une méthode de chainage adaptée aux similarités locales, *un nouveau type de chaînage colinéaire, permettant des chevauchements proportionnels*. Nous avons donné une formulation de ce nouveau problème et réalisé un premier algorithme. L'algorithme, qui adopte une approche de programmation dynamique basée sur le paradigme de la "sweep-line", donne une solution optimale, *i.e.* est exacte, et s'exécute en temps quadratique. Nous avons montré que cet algorithme, comparé au chainage colinéaire classique, améliore les résultats sur des génomes bactériens, tout en restant aussi efficace en pratique. Notre nouvel outil, YOC, a été évalué ensemble avec quatre autres outils d'alignement sur un ensemble de données composé de 694 couples de génomes bactériens intra-espèces. Les résultats montrent que YOC améliore les cas divergents en détectant des similarités plus distantes et en évitant les régions mal alignées. En conclusion, YOC semble être plus facile à appliquer de manière automatique et systématique, parce qu'il nécessite pas un post-traitement des régions mal alignées, ni un paramétrage complexe.

Mots-clés: génomique comparative, alignement des génomes complets, stratégie basée sur des ancres, graines espacées, chaînage des fragments, programmation dynamique, graphe trapézoïdal

Remerciements

Quand j'ai commencé ma thèse, je ne m'étais pas rendue compte de l'ampleur et de la difficulté de cette aventure. Je n'imaginais pas à quel point elle allait m'influencer, sur le plan professionnel comme sur le plan personnel. Heureusement plusieurs personnes m'ont épaulée durant ces années. Voici enfin venue pour moi l'occasion de les remercier.

Je tiens tout d'abord à remercier Eric Rivals qui fut un parfait directeur de thèse. Par son enthousiasme sans faille, sa persévérance, son engagement professionnel et surtout son humanité, il m'a toujours aidée à aller de l'avant. Même dans les moments les plus difficiles quand on se trouvait bloqués dans nos idées, nos doutes et nos inquiétudes, il a su trouver les mots justes pour me donner la force de continuer et, souvent, de recommencer. Ses vastes connaissances, sa rigueur de raisonnement, sa manière courageuse car pas toujours conventionnelle d'aborder les problèmes scientifiques et ses capacités rédactionnelles seront toujours pour moi un excellent exemple de chemin à suivre. Enfin, je tiens tout particulièrement le remercier, ainsi qu'à sa compagne Frederike, pour m'avoir accueillie à mon arrivée, ainsi que pour leur aide et leur amitié dans les moments compliqués de mes débuts à Montpellier.

Je voudrais ensuite remercier à Hélène Chiapello qui a été pour moi une co-encadrante exemplaire, et aussi une vraie amie. Merci à elle d'avoir pris de son temps pour "mettre la main à la pâte" avec moi pour faire avancer les choses, d'avoir cru dans nos idées et de les avoir défendues sans cesse même quand nous on y croyait plus. Je remercie également chaleureusement Célia Michotey, pour avoir essuyé tant de difficultés avec moi et pour avoir été toujours là quand j'avais besoin d'elle. Il va sans dire que sans leur aide précieuse, cette thèse n'aurait jamais vu le jour.

Je tiens à adresser un merci tout particulier aux membres de mon jury de thèse. Les rapporteurs, Marie-France Sagot et Mathieu Blanchette, pour leurs remarques judicieuses et leur efficacité à assurer les délais très serrés de ma soutenance. Thérèse Commes, pour avoir accepté de présider mon jury de thèse et pour m'avoir tant aidé le jour de la soutenance par sa gentillesse et son humour. Thomas Faraut et Vincent Ranwez, pour avoir accepté d'examiner ma thèse. Et enfin Gilles Caraux pour m'avoir fait l'honneur de remplacer mon du directeur de thèse lors de ma soutenance. Il s'est acquitté de cette tâche d'une manière tellement exemplaire qu'il m'a fait pleurer de bonheur.

Ensuite, je voudrais remercier bien d'autres personnes du LIRMM, membres de l'équipe MAB ou amis, qui ont contribué par leur écoute, leurs conseils et encouragements au bon déroulement de ma thèse. Je remercie plus particulièrement Alban Mancheron, Laurent Bréhélin et Laurent Noé avec lesquels j'ai eu la chance de collaborer. Je remercie Henri Vial pour sa bonne volonté, ainsi que Annie Chateau,

Sèverine Bérard et Vincent Lefort pour leur support constant et leur amitié. Enfin, je remercie à tous les membres de l'équipe avec lesquels j'ai eu l'occasion de partager des jolis moments, que ça soit pendant nos pauses improvisées, nos pique-niques, nos repas, ou nos soirées à "Couleurs de bières".

Ces années de thèse n'ont pas été liées seulement à la recherche, mais aussi à l'enseignement. Je voudrais donc remercier les différents enseignants qui m'ont fait confiance depuis le début, malgré la barrière de la langue. Merci donc Thérèse Libourel, Isabelle Mougenot, Mathieu Roche, Jean-François Vilarem, Michel Leclère, Annie Chateau, Sèverine Bérard, Vincent Boudet et Christophe Dony. Grâce à vous cette épreuve m'a paru facile à surmonter.

Au niveau du LIRMM, je voudrais remercier les membres du service d'accueil et du service administratif, et tout particulièrement Elisabeth Greverie, Cécile Lukasik, Elisabeth Petiot et Caroline Ycre qui m'ont constamment aidé et qui ont toujours répondu à mes questions.

Un grand merci aussi à mes amis doctorants et post-doctorants avec lesquels j'ai partagé cette longue tranche de vie. Je remercie Amel, on ne peut pas rêver d'une copine plus gentille, plus à l'écoute et plus adorable. Ben, sans son humour, son ironie et son amitié inconditionnelle, certains jours auraient été cent fois plus longs. Mathieu, pour tous les beaux moments qu'on a passé ensemble et ... "pour avoir contribué d'une manière ou d'une autre à la réalisation de cette thèse". Floréal, un merci spécial pour nos discussions interminables :-) et son aide toujours précieuse. Oliver G. et Fred, les deux bonhommes que j'adore et sans lesquels ma vie à Montpellier aurait été si fade au début. Cécile et Céline, pour avoir partagé avec moi ce fameux bureau des filles. Fabio et Jean Philippe, pour les fous rires journaliers. Olivier M. et Jean-Baka, pour leur bonne humeur éternelle et leur amitié. Merci aussi à Nico P, Nico T, Alexis, Sam, Guillaume, Amine, Seb, Gilles, Julien, Xavier, Fabien, Zeina et Fady.

Sur un plan personnel, je remercie ma famille et tout spécialement mes parents qui m'ont tant soutenu et qui ont tout donné pour que j'arrive jusqu'ici. Sans eux, rien de tout cela n'aurait été possible. Ensuite je voudrais remercier à ceux qui ont été ma deuxième famille pendant toutes ces années: Jean, Christian et Nadine, et tout particulièrement Marité. J'aimerais aussi remercier mes amis en dehors du monde de la recherche. Magi, pour ses encouragements et son soutien. Greg, sans lequel les moments les plus durs de ma dernière année de thèse auraient été insurmontables. Enfin, je remercie mon petit chéri, Jean-Rémy, pour sa patience et sa gentillesse sans limites, pour son aide constante, pour tous ses conseils, pour son soutien et son amour inconditionnel.

Contents

Introduction

Context The unprecedented sequencing capacity offered by High Throughput Sequencing technologies allows rapid whole genome sequencing at low costs. With more than 1000 complete genomes available for bacteria, archea, and eukarya, comparative genomic analysis become essential for the functional annotation of genomes, the understanding of their structure and evolution, and for phylogenomics (Bigot et al., 2005; Delsuc et al., 2005). Roughly speaking, comparative genomic analysis aim at identifying the shared or specific genome parts of individuals, strains, species. These parts may bear different names depending on the context, *e.g.* backbone and variable segments for bacterial strains (Chiapello et al., 2008).

Many projects aim at sequencing several genomes of a species or of a genus to acquire functional and evolutionary knowledge from the genomic variability, among which the 1000 human genomes project (`http://www.1000genomes.org`) or the Microbial Genome Program of U.S. Department of Energy (`http://microbialgenomics.energy.gov`). With such genomic data at hand, comparative genomic analysis are used in order to annotate genes and infer their homology/orthology relationships. Further more, comparative analysis also unravelled the forces driving genome diversity and evolution, *e.g.* (Leclercq et al., 2010). Similarly, the comparison of three strains of the ruminants' pathogen, *E. ruminantium*, underlined the importance of tandem duplication as a source of diversity and annotated some strain specific genes (Frutos et al., 2006). They also served to identify species specific genes that could explain the increased capacity to cause disease in *Candida albicans* compared to its closest relative, *Candida dubliniensis* (Jackson et al., 2009). Comparative analysis represent an important step in post-genomic vaccine design (Serruto and Rappuoli, 2006; Serruto et al., 2009), in which shared and variable genes of multiple strains are selected for further immunisation tests and may lead to design broadly protective vaccines (Maione et al., 2005).

The two mostly used approaches for comparative genomics rely on different information levels. The first involves comparing the proteomes, as sets of proteins, to predict orthology (Tatusov et al., 1997). This requires the proteins to be correctly annotated. The second solution is *whole genome alignment (WGA)* at the nucleotide level.

Despite the number of dedicated studies during the last years, see (Miller et al., 2004; Blanchette, 2007) for reviews, whole genome alignments are still in an exploratory phase. Moreover, the vast majority of the sequenced genomes are coming from bacteria, due to their short sequences and to their high economical and medical interest. However, even with a WGA at hand, it is not straightforward to obtain the segmentation of bacterial strains in backbone and variable segments (Chiapello et al., 2008). Hence, reliable tools for WGA in particular for bacterial genomes, still need to be

designed.

Moreover, only few methods were established for estimating the quality of a WGA. Recently, two studies dedicated to the precise examination of the quality of alignments (Prakash and Tompa, 2007; Lunter et al., 2008), identified several types of alignment errors and estimated, based on statistical methods, the amount of the alignment being suspiciously aligned. Compared to (Prakash and Tompa, 2007), (Lunter et al., 2008) focuses on homology at the nucleotide level in pairwise alignments, based on the assumption that the regions of homology are correctly assigned. In (Prakash and Tompa, 2007) on the other hand, the accuracy at each aligned site in multiple alignments is evaluated and alignment regions are classified into well aligned and suspiciously aligned.

Existing methods Classical alignment solutions, based on the dynamic programming strategy, cannot be adapted to the complex case of aligning whole genomes, because of the lengths and the levels of divergence and shuffling of genomic sequences. Therefore, WGA tools are heuristics and they are generally based on a complex strategy composed of several phases, *i.e. anchor based strategy* (Delcher et al., 1999; Brudno et al., 2003b; Hohl et al., 2002; Darling et al., 2004; Treangen and Messeguer, 2006; Darling et al., 2010).

In the first phase of the anchor based strategy, methods look for *sequence similarities*, *i.e.* fragments. Then, these similarities are filtered with a *chaining phase*, *i.e.* that builds a highest scoring chain of fragments, with overlaps between adjacent fragments being forbidden. Fragments selected in the chain are used as anchor points for the final alignment. Next, the first two phases are recursively applied, and finally a more sensitive method aligns those regions that are left unaligned between the identified anchor points.

For computing fragments in the first phase of the strategy, methods searching for exact or approximate matches are employed. In the second phase, corresponding to the chaining of fragments, two types of approaches exist: the classical method, (i) computing a collinear chain or (ii) computing a chain without imposing the collinearity constraint, *i.e.* allowing for rearrangements in the chain. If several dynamic programming solutions exist for the collinear chaining problem (Felsner et al., 1995; Abouelhoda and Ohlebusch, 2005), for the problem of chaining with rearrangements heuristics are used, as the problem was proved to be NP-complete in (Bafna et al., 1996). Not dealing with overlaps is also an important drawback of chaining methods. In fact, several methodological and biological causes for overlaps can be identified. Small overlaps are often caused by equality over a few base pairs of fragment ends due to randomness, since the alphabet has only four letters. To handle such cases, one could set a constant, large enough, maximal allowed overlap threshold. However, biological structures like tandem repeats (TR) that vary in number of copy units generate overlaps that are large relatively to the fragments involved. Such cases cannot be solved by fixing maximal length overlaps: only proportional overlaps can handle these. As variable-number tandem repeats occur in genomes of numerous species coming from all kingdoms of

life, the problem of dealing with proportional overlaps is of great importance. In the collinear case, algorithms can be extended to handle fixed length overlaps between adjacent fragments, however for proportional overlaps no method was proposed up to now.

With the flood of genome alignment tools in the last few years, one needs to compare the quality of alignments between different tools in order to identify their limitations and to be able to choose the best suited method in each specific situation. Unfortunately, very little work has been done on designing benchmarks, comparison protocols and biological evaluation criteria for this task, which should allow systematic analysis of WGA results. Two very recent studies (Swidan and Shamir, 2009; Chen and Tompa, 2010) carried out assessments of the quality of WGA on bacteria, respectively vertebrates, and came to a general similar conclusion: *building accurate WGA remains a challenge*. The study presented in (Chen and Tompa, 2010), based on 28 vertebrate genomes, concludes that there is a lack of general agreement between alignment methods, *e.g.* even for the well-known mouse genome. Moreover, they show that building accurate multiple WGA remains an important challenge for non-coding regions and distantly related species. Compared to (Chen and Tompa, 2010) who analyzed the complete alignment at the nucleotide level, the goal of (Swidan and Shamir, 2009) was to evaluate the quality of the segmentation of bacterial genomes with respect to their mosaic structure, and to quantitatively assess the quality of the alignment. They revealed that closely-related bacteria often have highly divergent gene content and therefore WGA tools encounter unexpected difficulty in these cases.

Moreover, as most WGA tools are variations on the same strategy, studies addressing the global evaluation of this strategy and the impact of each individual phase on the final result, are needed.

Contributions In this manuscript we address several of the research directions mentioned above. First, we design a protocol for evaluating the correctness and performance of WGA tools from both computational and biological view-points.

Second, based on an initial evaluation of existing tools on pairs of intra-species bacterial genomes, we propose to improve the anchor based strategy by using, for the first time in the first phase of the strategy for WGA, local similarities instead of short matches. A preliminary evaluation of the impact of this first phase on the WGA results, reveals the need of a chaining method adapted to local similarities, allowing for overlaps.

We thus introduce a novel type of collinear chaining: allowing for proportional overlaps. We give the first algorithm for this problem and show its impact on real bacterial data, compared to classical collinear chaining.

Finally, we obtain a novel pairwise WGA tool, YOC, composed of only two phases, unlike classical, four phase WGA tools: local similarities & chaining with overlaps. This novel tool is evaluated together with other four WGA tools on a dataset composed of 694 pairs of intra-species bacterial genomes, based on the protocol that we designed.

Outline of the manuscript This manuscript has an introduction, a brief conclusion and is organised in six additional chapters as follows.

Chapter 1 is a state of the art for classical sequence alignment. It begins with an introduction to the alignment of biological sequences, while mentioning some of the applications and the open research directions in the field. The second section is dedicated to the subject of mutations, whose discovery is one of the targets of sequence comparison methods. It is followed by several sections addressing the classical solutions for pairwise and multiple sequence alignment and their scoring models. Finally, in the last section, we address the problem of evaluating alignments, meaning existing benchmarks, simulation approaches, and the definition of adapted comparison measures and scores.

Chapter 2 gives an overview of the whole genome alignment problem. First, applications, limitations and further research directions are presented in the introduction of this chapter. Next, we describe the classical method for WGA: *the anchor based strategy*. We dedicate the following section to the methods implemented in the first phase of this strategy. Next, we describe the most commonly used WGA tools for both pairwise and multiple WGA, in both collinear and rearranged cases. Finally, we discuss existing work on estimating the quality of alignment results and on doing comparative assessments of WGA tools.

In Chapter 3 we present our first contributions to the whole genome alignment field. In the first section, we describe a protocol for evaluating the correctness and performance of WGA tools from a computational point of view. Second, we do a preliminary evaluation of alignment results given by two state-of-art WGA tools: MGA and Mauve on pairs of intra-species bacterial genomes. Next, we propose to improve the anchor based strategy by using local similarities (LS) instead of short matches in the first phase of the strategy. More precisely, we suggest to exploit programs that detect LS based on spaced seeds, which prove to be both sensitive and efficient. Based on these results we introduce a novel definition for chaining, allowing for overlaps. This new problem is also addressed in this thesis and is the topic of Chapter 5.

Chapter 4 is an overview of the chaining problem. It begins by defining the classical *collinear chaining problem*, followed by a straightforward, quadratic, dynamic programming algorithm solving it. Next, we give a short overview of the techniques employed by chaining methods in order to speed up the solutions. We then give an efficient solution for the collinear chaining problem in a formulation with *k-trapezoid graphs* that can be adapted to the one-dimensional case too. Several other solutions are briefly described in the following sections. In the end of the chapter, we address two generalizations of the collinear chaining problem: *glocal chaining* and *chaining with rearrangements*.

In Chapter 5 we give our contribution to the fragment chaining problem, *i.e.* introduce a new problem and give the first exact algorithms for it. We start by defining the *novel collinear chaining problem with proportional overlaps*. In this definition, we allow for proportional overlaps between anchors, and seek to optimize a criterion related to the coverage but without counting twice overlapping regions. We then give a formulation for this problem and show that this problem can be solved naturally

by a dynamic programming approach, which is not trivial at first glance. This yields a fully quadratic dynamic programming algorithm, which sometimes requires unreasonable running times for real case applications. Following the work by Felsner and colleagues, we exploited a box representation of the set of genomic fragments, which allows to adopt a sweep line paradigm for this problem. We thus exhibited another dynamic programming algorithm that avoids computing certain dependencies for the sake of efficiency, like sparse dynamic programming techniques. Although quadratic, this sweep line algorithm turns out to be sufficiently efficient to compute a chain by selecting among millions of input fragments, as shown by our comparisons. All 694 cases of pairwise bacterial strains comparisons were processed with this algorithm, and its improvement compared to a chain without overlaps showed a real advantage.

In Chapter 6 we continue the experiments that we started in Chapter 3, this time by combining local similarities with the novel chaining algorithm with overlaps. We thus obtain a novel pairwise WGA tool, YOC, based on a simplified two-phase strategy. We first compare YOC to four state-of-the-art WGA tools: MGA, Mauve, LAGAN and ProgressiveMauve from a computational point of view, based on the computational criteria defined in the previous experimentation chapter. Finally, using biological criteria based on orthologous genes, we give a preliminary biological analysis of the alignment results.

The thesis concludes with a brief overview of the contributions and perspectives.

List of publications (updated in June 2011**)**

- international journals:
  - *Novel definition and algorithm for chaining fragments with proportional overlaps*
    R. Uricaru, A. Mancheron, E. Rivals
    To appear in Journal of Computational Biology.

  - *An alternative approach to multiple genome comparison*
    A. Mancheron, **R. Uricaru**, E. Rivals
    Nucleic Acids Research (NAR), 2011.

  - *Reliable bacterial genome comparison tools*
    E. Rivals, A. Mancheron, **R. Uricaru**
    ERCIM News, Vol. 82, p. 17–18, 2010.

- international conferences:
  - *Novel definition and algorithm for chaining fragments with proportional overlaps*
    R. Uricaru, A. Mancheron, E. Rivals
    RECOMB Comparative Genomics, Lecture Notes in Computer Science (LNCS), Springer-Verlag, Vol. 6398, p. 161–172, 2010.

- national conferences:
  - *Improved sensitivity and reliability of anchor based genome alignment*
    R. Uricaru, C. Michotey, L. Noé, H. Chiapello, and E. Rivals.
    Actes des Journes Ouvertes Biologie Informatique Mathmatiques (JOBIM), p. 31–36, 2009.

- international posters:
  - *An alternative approach to multiple genome comparison*
    A. Mancheron, **R. Uricaru**, E. Rivals
    Human Genome Meeting, Montpellier, 2010.
  - *A novel approach for comparative genomics & annotation transfer*
    A. Mancheron, **R. Uricaru**, E. Rivals
    RECOMB Comparative Genomics, Ottawa, 2010.

1 State-of-the-art. The basics of sequence alignment

Despite all the progress that has been made in sequence alignment, there are still many challenges to face. Amongst them, we mention (i) improving existing methods for aligning highly divergent genomes, (ii) developing better methods for aligning larger datasets, both more and longer sequences, particularly at the scale of complete genomes, (iii) designing objective functions and parameters leading to reliable homology predictions, (iv) developing benchmarks for testing alignment algorithms, (v) exploring the effects of alignment error on bioinformatics analysis.

Contents

1.1 Introduction to sequence alignment

Sequence alignment is a fundamental procedure in any biological study, comparing two, *i.e.* *pairwise alignment*, or more biological sequences, *i.e.* *multiple alignment* (MSA), whether DNA, RNA, or protein. The basic sequence alignment problem, pairwise alignment, is in fact a particular case of the *string alignment* problem, *i.e.* a way to display a transformation of one string to another.

A naive description of this task would be that aligning two strings is done by inserting spaces, *i.e.* *gaps*, either inside the strings, *i.e.* in between the characters, or to their ends, so that when placing strings one above the other, identical or similar characters are aligned. Starting from this description, one may imagine a text format to represent sequence alignments with aligned columns containing identical or similar characters indicated with a system of symbols, see Figure 1.1 for such a pairwise alignment representation of two protein sequences.

```
HBA_HUMAN    GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHK
             G+ +VK+HGKKV A+++++AH+D++ +++++LS+LH K
HBB_HUMAN    GNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDK
```

Figure 1.1: Pairwise sequence alignment of *human alpha* and *beta globin* (SWISS-PROT database identifiers HBA_HUMAN and HBB_HUMAN), showing the obvious similarity between the two amino-acid sequences. The central line in the alignment indicates identical positions with letters and similar positions with the + sign, *i.e.* pairs of residues having a positive score in the substitution matrix used to score the alignment.

Many sequence visualization programs also use colour to display information about the properties of the individual sequence elements, see Figure 1.5 for a multiple alignment representation. For multiple alignments, the last row is usually the *consensus sequence* determined by the alignment, represented with a sequence logo in which the size of each nucleotide or amino acid letter corresponds to its degree of conservation in the column.

The *edit distance* is an alternate way of presenting the transformation of one string into the other and the set of operations needed for this transformation is called an *edit transcript*. From a mathematical point of view, alignments and edit transcripts are equivalent, but an alignment alone hides the mutational process. In fact, different evolutionary models are described by different permitted operations and yet those can result in the same alignment. As they make no statement on the process, alignments are usually preferred when comparing biological sequences (Gusfield, 1997).

The stated goal of biological sequence alignment is inferring which sites within sequences are *homologous*, meaning that they share a common evolutionary history. This is not an easy task, as identity can be truly due to homology, but may often be

due to substitutions and wrong residue pairing, *i.e.* misalignment. Thus, comparing two or more biological sequences means much more than looking for a plausible alignment between them. When aligning biological sequences one searches for probabilistic evidence that they evolved from a common ancestor, *i.e.* one is looking for the alignment that is most probable from a biological point of view. Alignment algorithms try to reach this biological goal by doing an efficient optimization of an objective function. One should keep in mind the fact that a solution being computationally optimal is not necessarily biologically correct. This is one observation that can be made for any computational biology task and not just for alignment. For example, it has been shown through simulation (Kumar, 1996; Takahashi and Nei, 2000) that the true phylogenetic tree is often not the optimal one for a given dataset.

One should consider distinctly *pairwise* and *multiple sequence alignment* (MSA), *i.e.* an alignment of three or more sequences that are usually assumed to have an evolutionary relationship and descend from a common ancestor.

Pairwise sequence alignment methods are used to find the best-matching pairs of sequence pieces, *i.e.* *local*, or a correspondence between the characters on the entire length of the sequences, *i.e.* *global alignments*, of two sequences, see Section 1.6.1. Pairwise alignments are efficient to calculate and are often used for searching a database for sequences with high similarity to a query, or as a basic step of multiple sequence alignment methods. For both global and local pairwise alignment multiple solutions exist: from techniques that compute optimal alignments with *dynamic programming*, to involved heuristics adapted to longer, more divergent sequences, see Section 1.6.1.

Multiple sequence alignments (MSA) methods MSA are among the most useful objects in bioinformatics. Usually when speaking about MSA, one refers to global multiple alignment. The resulting MSAs can thus be used to infer sequence homology and conduct a phylogenetic analysis in order to assess the shared evolutionary origins of the sequences, to detect regions of variability or conservation and to build profiles, *i.e.* probabilistic models, for families of proteins. MSA requires more sophisticated methodologies than pairwise alignment. Due to its computational complexity, most multiple sequence alignment programs use heuristic methods rather than global optimization because identifying the optimal alignment between more than a few sequences of moderate length is prohibitive, see Section 1.6.2.

Scoring alignments An important point for the sequence alignment problem is to choose between several potential alignments the one that is closer to the biologically correct alignment. For this, one needs to compute a score that reflects the quality of each alignment and that should be easily calculated. Formulas behind scores go from simple additive scores to complex maximum likelihood values. Whatever the score formula one uses, one has to estimate the apparition rate of each basic mutational event: substitutions, insertions and deletions, rates that may differ a lot between different

species, different sequences and in different regions of the same sequenc. Therefore, sequence alignment is done by *first*, choosing a scoring model to rank alignments, see Section 1.3, *second*, aligning the sequences with an algorithm based on an optimization procedure during which the alignment score is computed, see Section 1.6 and *finally*, evaluating the significance of the scores.

Comparison of alignments Facing the growing number of alignment tools, one needs to compare their results, in order to determine the performance of methods under different conditions. This should guide users in choosing the most appropriate program for a given problem. For this, one needs benchmarks of real or simulated data and comparison measures, see Section 1.7.

Applications There is a certain belief nowadays, namely that sequence alignment is an easy and fast task and thus, that it has become a trivial subject in bioinformatics (Rosenberg, 2009, chap. 1). In fact, sequence alignment as an area of research is now an underappreciated aspect of comparative genomics, mostly because alignment tools have become so numerous, reliable and fast. It is however a vital step when studying deeper questions such as the identification and quantification of conserved regions or functional motifs (Kirkness et al., 2003; Thomas et al., 2003b), profiling of genetic disease (Miller and Kumar, 2001; Miller et al., 2003), phylogenetic analysis (Felsenstein, 1981; Felsenstein, 2004), ancestral sequence profiling and prediction (Cai et al., 2004; Hall, 2006), secondary structure identification (Coventry et al., 2004; Dowell and Eddy, 2004; Holmes, 2005; Knudsen and Hein, 1999), noncoding functional RNA detection (Di Bernardo et al., 2003; Rivas and Eddy, 2001).

Future work on sequence alignment A second opinion on the sequence alignment subject makes surface lately: *sequence alignment should not be taken for granted* (Rosenberg, 2009), as there are still many challenges and issues that need to be overcome, see details below.

1. The overwhelming number of tools released over the last twenty years are a proof of the fact that there is no tool that can be used in any situation only by tuning the parameters. In fact, anyone working with sequence alignment asked himself **which program to use** and understood that there is no simple answer to this question. The best alignment program depends on several factors for a given situation: the purpose of building an alignment and the nature of the data to be aligned, *i.e.* the type of the sequences (*e.g.*, DNA, RNA, protein), the number of sequences, their lengths and their evolutionary divergence. Thus, an informed user has to aware of the trade-off between the simplicity of use of a software and the necessity of fixing a big number of parameters in order to make the best of use in a given situation.

2. Moreover one may be submerged by the number of different alignment results that obtained with different tools and choosing among them is not an easy task

in itself. Therefore, **benchmarks and evaluation measures** for sequence alignment are another important research directions, beside designing alignment tools. In fact, benchmarks are vital in order to assure high quality tools, to identify their strong and weak points, to measure the improvements introduced by new methods and to enable users to choose among tools. As we shall see in Section 1.7, such benchmarks exist mostly for protein sequences, while for non-coding DNA sequences simulations are usually preferred. As researchers started to show increasing interest to non-coding DNA sequences, a real need for evaluation of such alignments appeared.

3. Increasing evidence is now indicating that what was before considered as **junk DNA**, *i.e.* non-coding DNA (*e.g.*, more than 98% of the human genome), is not *junk* at all. In fact it has been found to have various regulatory roles, meaning that non-coding DNA influences the behaviour of the genes, the coding DNA, in important ways, *e.g.* it may work as a genetic "switch" regulating when and where genes are expressed (Carroll et al., 2008), it may contain features essential to chromosome structure, centromere function and homolog recognition (Subirana and Messeguer, 2010), or even disease-causing genetic variants (Cobb et al., 2008). Despite all these advancements, the relationship between non-coding DNA and the DNA of genes remains globally a mystery.

4. Very recently, **Next Generational Sequencing (NGS)** technologies, capable of sequencing DNA at unprecedented speed, enable impressive scientific achievements and novel biological applications (Schuster, 2007). The development of NGS is forcing a reconsideration of the computational methods used for genome analysis, with the problems of read mapping and genome assembly becoming much more complex. Simultaneously, NGS addresses problems previously not addressed with genome sequencing, such as the prediction of structural or copy number polymorphisms. Moreover, the NGS data has a very different error model, requiring modifications to classical algorithms, and the sheer size of the data requires the use of effective algorithms, appropriate hardware, and effective implementations. See (Metzker, 2009) for a review on NGS methodological needs and applications.

5. A relatively novel problem in sequence alignment is aligning **whole genomic sequences**, see Chapter 2 and Chapter 3 that are entirely dedicated to this problem.

This chapter is organized as follows: in Section 1.2 we shortly describe genome dynamics, in Section 1.3 we address the problem of scoring alignments, in Section 1.4 we discuss the differences between the global and local alignment problems, in Section 1.6.1 we present several pairwise alignment methods, Section 1.6.2 is dedicated to multiple alignment tools, and finally, in Section 1.7 we discuss the evaluation of alignment results.

1.2 Genome dynamics: from punctual mutations to large-scale changes

Most new genes and proteins will evolve through changes in the DNA sequence of existing genes and proteins, *i.e. mutations*. If one scores the differences occurring between the sequences of related organisms, one can judge the distance between them.

Mutations can be caused by radiation, viruses, transposons and mutagenic chemicals, as well as by errors occurring during meiosis or DNA replication. They can determine different types of changes in the DNA sequence that can either have no effect, alter the product of a gene, or prevent the gene from functioning properly. Due to the damaging effects that mutations may have on cells, organisms developed mechanisms such as DNA repair in order to remove them. Mutation rates vary across species, being even beneficial in rare situations as they allow organisms to adapt more quickly to the environment and they may even propagate throughout the population.

One can classify mutations as follows:

- small-scale mutations affecting one or a few sites
  - point mutations often caused by chemicals or malfunction of DNA replication, exchange a single character for another. For DNA sequences these changes are classified as *transitions* (purine for purine $A \longleftrightarrow G$ or pyrimidine for pyrimidine $C \longleftrightarrow T$) or *transversions*. Point mutations occurring within the protein coding region of a gene may be classified into three kinds, depending upon what the erroneous codon codes for: *silent mutations* (coding for the same amino acid), *missense mutations* (coding for a different amino acid) and *noncoding mutations* (coding for a stop and thus truncating the protein).
  - *insertions* meaning adding extra characters. If added to the coding region of a gene, they may alter splicing of the mRNA or cause a shift in the reading frame (frameshift), both of which can alter the gene product.
  - *deletions* remove one or more characters, and as the insertions, deletions can alter the reading frame of the gene. We should note that even though insertions and deletions are two completely different mechanisms, from the computational point of view of the sequence alignment task, they are often treated in the same way, grouped together and referred to as *indels*. This is due to the lack of information on the common ancestral sequence of the aligned sequences.
- large-scale mutations in chromosomal structure, i.e. *genomic rearrangements*
  - *duplications*, leading to multiple copies of chromosomal regions;
  - *deletions* of large chromosomal regions, leading to the loss of the genes within these regions;

– juxtaposition of previously separate pieces of DNA, potentially bringing together separate genes to form functionally distinct fusion-genes, like: *translocations* (interchange of genetic parts from two chromosomes), *transpositions* (interchange of genetic parts in a chromosome), *interstitial deletions* (removing a DNA segment from a single chromosome thus apposing previously distant genes), *chromosomal inversions* (reversing the orientation of a chromosomal segment), *fission* (breakage of one chromosome in two) and *fusion* of two chromosomes (joining them into one).

The efficiency of the alignment of sequences can be increased by taking in account the type of sequence in which the mutation is occurring and the likelihood that such a mutation occurs in that context. Thus, small-scale mutations hardens the classical sequence alignment task (small-scale genomic sequences) and can be taken into account in the scoring model, see Section 1.3. On the other hand, large-scale mutations determine genomic *rearrangements* that need to be taken into account in whole genome alignment, as we shall see in Chapter 2.

1.3 Scoring models

Most sequence alignment programs implement scoring functions to compute an overall score of an alignment. The goal is to obtain enough evidence, *i.e.* scores high enough, that they diverged from a common ancestor by several mutation and selection events. The basic mutational events that are considered are *substitutions*, *insertions* and *deletions* (indels) referred together as gaps in the scoring scheme.

No perfect model for scoring biological sequence alignments exists. Varying situations need appropriate scoring schemes. In the case of *pairwise alignment*, the most common scoring models are based on an *additive approach* that needs to specify scores for every possible way in which pairs of sites can be compared, see Section 1.3.1 for a complete description of this kind of approaches. In order to use an additive scoring scheme one has to make the assumption that mutations at different sites in a sequence have occurred independently. This appears to be a reasonable approximation for DNA and protein sequences.

Once the optimal alignments have been found, the question is how to assess the significance of its score, *i.e.* whether it is meaningful from a biological sense by giving enough evidence for a homology, or it is just the best alignment between two unrelated sequences. For this, it is necessary to be very precise regarding the thresholds. In fact, appropriate thresholds depend on the lengths of the sequences and the size of the database containing the sequences. Moreover, raw score thresholds are affected by the choice of parameters in the scoring function.

For assessing scores, two different approaches can be used: a *Bayesian method* based on the comparison of different models and the classical method, namely the *extreme value distribution* (EVD). The EVD calculates the chance of a match score greater than the observed value, assuming a null model, which in this case is that the sequences are unrelated. In (Karlin and Altschul, 1990) the appropriate EVD distribution for

ungapped alignments was obtained analytically. In (Mott, 1992) it was suggested that gapped alignment scores follow the same form of extreme value distribution as ungapped scores, but for the moment there is no analytical theory sustaining this observation.

1.3.1 Pairwise sequence alignment

When scoring pairwise biological sequence alignments, three different scores for three different types of operations must be defined:

- the cost of aligning a pair of sites containing the same character in both sequences, *i.e. match*, usually corresponding to a positive value called a *match bonus*;
- the cost of aligning a pair of sites containing different characters in the sequences, *i.e. mismatch*, corresponding to a substitution score;
- the cost of aligning a character in one sequence to a gap in the second sequence, *i.e. indels*, corresponding to a gap penalty.

Thus, operations may be positive, *i.e.* bringing a positive score as for matches and some mismatches, or negative as for some mismatches and gaps, *i.e.* penalizing the total alignment score. The total score assigned to an alignment is the sum of terms for each aligned pair of residues plus the terms for the gaps. In this definition of scores, which represents a *similarity measure*, better alignments have higher scores. In order to find the best alignment, one needs to maximize the total score.

One must take into account the apparition rate of the basic mutational processes for a given type of sequences. For example, if we take mismatches, we know that they are not necessarily equal but they depend on the properties of the involved characters. Therefore scores are usually differentiated based on standard models of sequence evolution. This can be done by using appropriate *substitution matrices*, which set the match bonuses and the mismatch costs, as well as appropriate *gap penalties*, see below.

The substitution matrices that are usually used are empirically derived, i.e. for protein sequences PAM (Dayhoff et al., 1978), JTT (Jones et al., 1992), BLOSUM62 (Henikoff and Henikoff, 1992), OPTIMA (Kann et al., 2000) matrices. These matrices are based on biological factors such as the conservation, frequency, and evolutionary patterns of single amino-acids. Similarly, for DNA sequences, various substitution models give higher weights to transitions than to transversions. There exists no single best substitution matrix; one usually chooses a suitable matrix depending on the observed pairwise percentage of identity or divergence time.

Gaps and gaps penalties *Gaps consist of maximal consecutive runs of spaces in one sequence, facing a string of characters in the second sequence. They correspond to insertion or deletion events.* They may greatly affect the accuracy of the alignment and thus they must be treated with much care. Knowing the type of sequence in which the gap is occurring and the likelihood that a gap will occur in this context becomes crucial. First, gap penalties for different characters should be different. Second, gap penalties should be improved as to avoid treating them like identical single position events. Explicitly, we do not want the cost of a gap covering three positions to correspond to the triple the cost of a single gap, as it is the case for the *linear gap penalty*, *i.e.* $g \times l$ where g is the gap penalty and l the length of the gap.

One avoids this by using one cost for *opening a gap* and a second one for *extending* it, *i.e. affine gap penalty* (Altschul and Erickson, 1986; Gotoh, 1982; Gotoh, 1986; Taylor, 1984). Based on the empirical observation that long gaps are more likely to occur than several short gaps, in practice the opening of a gap is considered to cost more than extending it. Nowadays, most alignment tools use some form of affine gap cost. According to the affine scheme for the gap penalty, gaps are scored using the formula $o + e \times (l - 1)$, where o corresponds to the cost for opening a gap, e is a cost for extending it, and l the length of the gap. The example in Figure 1.2, extracted from (Rosenberg, 2009), shows a hypothetical alignment highlighting the advantage of the affine gap penalty scheme.

```
GATCGCGCGCGCGCGCATGC    GATCGCGCGCGCGCGCATGC
GATC--G--C--G--CATGC    GATCGCG-------CATGC
```

(a) (b)

Figure 1.2: Linear (a) versus affine gap penalties (b) in an alignment of two hypothetical DNA sequences. By using a linear gap penalty, all gaps are scored equally, which may result in an alignment with multiple short gaps. On the other hand, the affine gap penalties, involving a score for opening a gap and one for extending it, give a more accurate alignment. As opening a gap costs more than extending it, when using affine gap penalties, we obtain longer indel events, which are more likely than separate smaller gaps corresponding to multiple indel events.

The parameters corresponding to the affine gap penalties were first optimized by (Barton and Sternberg, 1987). Afterwards several groups have worked on improving these penalties (Altschul, 1998; Mott, 1999). To additionally refine the gap penalties, the context and the mechanisms of indel events must be thoroughly studied. By observing such patterns, gap penalties may be deduced empirically, as it is the case in the studies of (Reese and Pearson, 2002; Chang and Benner, 2004).

1.3.2 Multiple sequence alignment (MSA)

A *multiple sequence alignment* (MSA) is a sequence alignment of three or more biological sequences, generally protein or DNA. In many cases, the input set of query sequences are assumed to have an evolutionary relationship by which they share a lineage and are descended from a common ancestor.

In order to deal with the more complex problem of scoring *multiple alignments*, a scoring system should keep in mind the fact that sequences are not independent, but in fact they are related by a phylogenetic tree. However, even if we are given a correct phylogenetic tree for the sequences, the desired evolutionary model would still be very complex as probabilities of evolutionary changes would depend on the evolutionary times along branches in the tree, as well as on position specific constraints imposed by natural selection. Unfortunately, as one does not have enough data to parametrize such a model, simplifying assumptions must be made, like ignoring the phylogenetic tree or using position independent models.

Depending on whether the phylogenetic relationships among the sequences to align are provided or not, scoring systems can be classified into two categories. One category assumes the phylogenetic relations given, *i.e.* weighted sum-of-pairs (WSP) and maximum likelihood (ML), while the other does not, *i.e.* sum-of-pairs (SP), star or consensus scores and minimum entropy (ME).

As in the case of MSA there are multiple characters in each aligned column, deciding of the best way to score a column is not obvious for many different ways of scoring exist. The *sum-of-pairs* scoring scheme for example, scores each column of the alignment by summing the scores of all pairs of symbols in that column (based on a pairwise substitution matrix), and then sums over all column scores. The weakness of this method is that it tends to overweight the contributions to the score of many similar sequences. The *minimum entropy method* (ME) tries to minimize the entropy of each column, a convenient measure of the variability observed in an aligned column of residues. The more variable the column is, the higher the entropy, *i.e.* a completely conserved column would score 0. Thus, in this case, a good alignment minimizes the total entropy of the alignment.

1.4 Global versus local alignment

In the simplest case, both pairwise and multiple alignment methods make several a priori assumptions on the sequences under study, *i.e.* that:

- all input sequences are related by common structure, function and evolution, meaning that they are homologous;
- their biological relatedness is detectable at the primary sequence level;
- similarity extends over the entire length of the input sequences.

Under these assumptions, homologies in a set of sequences can be properly represented by a *global alignment*, which is an alignment extended from the beginning to the end of the sequences.

In reality, assuming that sequences are perfectly alignable on their entire length may be incorrect as a result of small and large scale mutations, and of genome shuffling when aligning large scale sequences, see Section 1.2. Thus only subsections of the sequences may be homologous and moreover, homologous subsections may be ordered differently. If one would try to compute a global alignment of the sequences, disregarding the rearrangements they have been undergone, at best the alignment obtained would contain a proper alignment of some true local homologies, surrounded by meaningless alignments of non related parts. In the worst case, alignments can even make no sense at all.

An alternative to global alignment is *local alignment*, which aligns subsections of the sequences regardless the overall global order within the sequence. Local alignment algorithms depend on a cutoff score, as only alignments that score above a threshold are accepted. Deciding on this cutoff can be difficult: if it set too low too many false positives will pass through and if it set too high some significant hits will be missed.

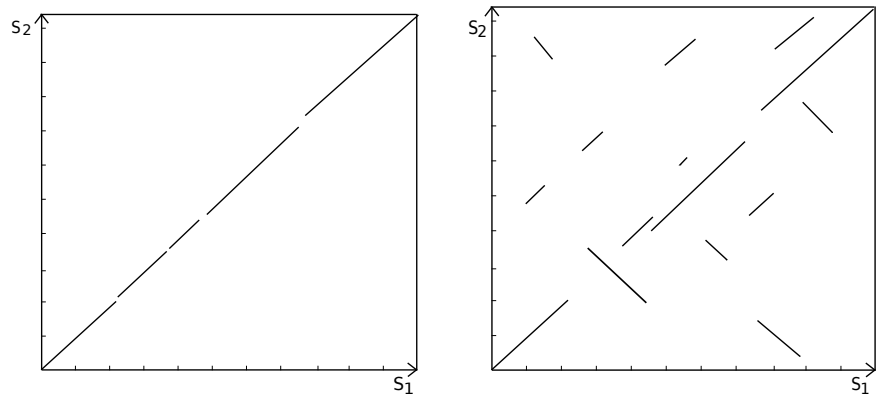
A technique called *dynamic programming* can be used to calculate an optimal global alignment efficiently (Needleman and Wunsch, 1970; Sellers, 1980; Durbin et al., 1998), as we shall see in Section 1.6. Interestingly, some small modifications of the Needleman & Wunsch (NW) algorithm are enough to solve the local alignment problem, which at first look seems a lot more complex than the global version (Smith and Waterman, 1981).

Moreover, dealing with large genomic sequences has become the ultimate goal in terms of sequence alignment due to the recent explosion of the number of sequenced genomes. Large sequences have the particularity that they are rarely conserved along their entire length. Therefore the alignment cannot be extended to the entire sequences, as conserved regions are usually interrupted by non conserved parts of the sequences. Global alignment is thus useless in this case.

Local alignment methods can be helpful but they also have some disadvantages. First, they do not suggest how the sequences could have evolved from their common ancestor. Second, in the case where sequences have n paralog copies of a particular feature, for the pairwise case, local aligners will return n^2 local alignments between all the pairs, whereas a global alignment reflects more clearly the evolutionary process. Therefore, the best solution in this case is to use a combination between local and global alignment, see Section 4.10 for a definition of alignment taking rearrangements into account, and Section 4.18 for a combination between global and local alignment called *glocal alignment*.

1.5 Visualize alignments

In order to visualize a pairwise sequence alignment, one usually uses a *dot plot*, *i.e.* similarity matrix. Dot plots are in fact the easiest and oldest way to visualize the



(a) (b)

Figure 1.3: Dot plots representing pairwise sequence alignments, where the first sequence corresponds to the horizontal axis and the second sequence to the vertical axis. (a) The alignment in the left image corresponds to a global alignment, as it covers the two sequences entirely and it follows closely the main diagonal of the plot. This suggests that the sequences were entirely homologous and that the homologous regions kept the same order on the two sequences. (b) The dot plot in the right image presents the case of two shuffled genomic sequences, *i.e.* whose homologous subsections are ordered differently. A global alignment makes no sense in this situation, thus a set of local alignments has been selected, suggesting the numerous mutational events that transformed the sequences.

similarity between two biological sequences. They correspond to two-dimensional matrices that have the sequences being compared along the vertical and horizontal axes. For a simple visual representation of the similarity between two sequences, individual cells in the matrix, *i.e.* representing pairs of positions in the two sequences, are shaded black if residues are identical, so that matching sequence segments appear as diagonal lines across the matrix. Insertions and deletions between sequences give rise to disruptions in this diagonal. Regions of local similarity or repetitive sequences give rise to further diagonal matches in addition to the central diagonal.

See Figure 1.3 for two examples of alignments presented as dot plots. If one uses a dot plot to represent a pairwise sequence alignment, then a perfect global alignment, *i.e.* due to identical sequences, corresponds to the complete main diagonal in the dot plot. Large scale alignments can also be represented by dot plots but for this, a zoom out of the similarity matrix needs to be done, meaning that a cell in the matrix corresponds to an entire subsequence and not to an unique residue; for large scale alignment representations, large scale rearrangements can be observed, however the small scale mutations are not visible at this level.

1.6 Alignment algorithms

In this section we discuss the different solutions for small-scale sequence alignment. First, one should understand that manually producing alignments is impossible even for the pairwise case, as generating all the potential alignments and choosing among them with a scoring function is an enormous task. In fact, the number of potential alignments is extremely high. For example, we shall take the simple case of a 100 characters sequence, aligned with a sequence composed of 95 characters. If all we do is add 5 gaps to the second sequence in order to obtain a total of 100 positions, there are approximately 55 million possible alignments (Krane and Raymer, 2003). As in reality we should probably add gaps to both sequences, the number of possible alignments becomes significantly greater. Thus, automated methods for aligning sequences are essential.

1.6.1 Pairwise alignment

Pairwise alignment is the basic task in the field of sequence alignment, see Figure 1.1 for an example of a pairwise alignment output. Below, we distinguish between global and local pairwise alignment and detail a classical dynamic programming global alignment solution and several heuristics for local alignment.

Alternate approaches to alignment use statistical estimation based on either maximum-likelihood or Bayesian methods. It was in the early 90s that important advances have been made in this direction, consisting in the formal development of hidden Markov models (HMMs) to describe the insertion-deletion process (Baldi et al., 1993; Baldi et al., 1994; Krogh and Brown, 1994; Eddy, 1995) for the sequence alignment problem. Additionally (Allison et al., 1992; Allison and Wallace, 1994) set the stage for Bayesian approaches.

Exact global alignment solutions. Needleman & Wunsch (NW) algorithm The first attempts of developing alignment methods for pairwise alignments were done in the mid 60s in (Fitch, 1966; Needleman and Blair, 1969) but an elegant solution was given by (Needleman and Wunsch, 1970). Their algorithm for two sequences, based on the dynamic programming technique and allowing gaps is still nowadays a reference on the matter. In (Gotoh, 1982) a more efficient version of this algorithm was proposed by decreasing the number of computational steps. Below, we describe a version of this technique for linear gap penalties; it can however be modified in order to deal with affine gaps. The algorithm finds a unique optimal alignment given an additive score even if several alignments may exist with the same score. Only few alignment tools output more than one alignment and the choice of this unique alignment is not very precise (Huang et al., 1990; Huang and Miller, 1991).

The main idea of the dynamic programming algorithm is to compose an optimal alignment of the two sequences S_1 of length n and S_2 of length m by using previous solutions for optimal alignments of smaller subsequences. For this, it builds a matrix M, with the value $M(i, j)$ being the score of the best alignment of the subsequences

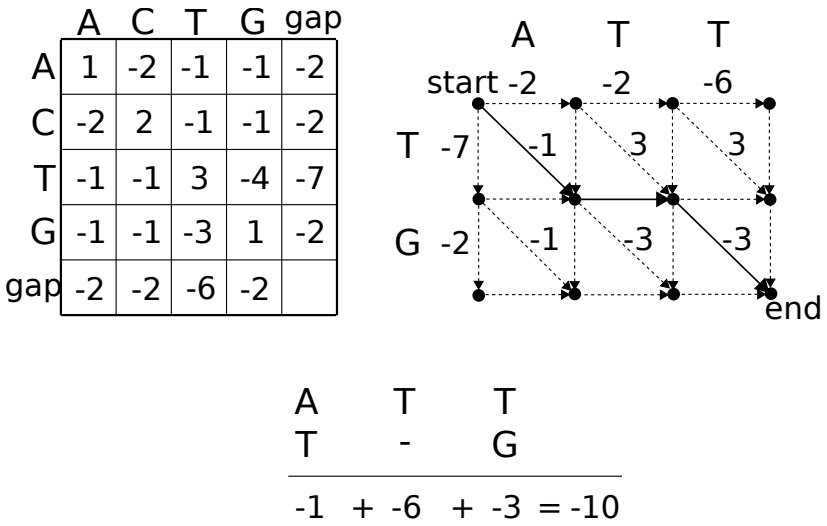


Figure 1.4: Two DNA sequences to be globally aligned, a substitution matrix (non symmetric) fixing the match and mismatch scores for each pair of characters, the corresponding gap penalties, and the resulted dynamic programming matrix obtained with the NW algorithm. Among the potential alignments of the two sequences, one was chosen as example.

$S_{1_{1\ldots i}}$ of S_1 and $S_{2_{1\ldots j}}$ of S_2. $M(i,j)$ can be computed recursively. At the beginning, $M(0,0)$ should be initialized with 0. Then, the matrix is filled from top left to bottom right, as once $M(i-i,j-1)$, $M(i-1,j)$ and $M(i,j-1)$ are known, $M(i,j)$ can be computed. If $s(S_{1_i}, S_{2_j})$ is the individual score of aligning the S_{1_i}, S_{2_j} pair of residues given by a substitution matrix, and g a gap cost, *i.e.* negative value, per inserted residue (in the general case, gap costs may differ for each residue), there are three possible ways of adding the last column to the alignment up to S_{1_i} and S_{2_j}:

- S_{1_i} aligned to S_{2_j}, $M(i,j) = M(i-1,j-1) + s(S_{1_i}, S_{2_j})$;
- S_{1_i} aligned to a gap, $M(i,j) = M(i-1,j) + g$
- S_{2_j} aligned to a gap, $M(i,j) = M(i,j-1) + g$

The best score up to S_{1_i}, S_{2_j} is the largest among these three options. This equation is applied repeatedly to fill in the matrix, by calculating the value in the bottom right corner of each square of four cells from one of the other three values (above left, left or above), see Figure 1.4. As cells are filled in, we also keep a pointer in each cell back to the cell from which $M(i,j)$ was derived. In order to complete the specification of the algorithm, the values $M(i,0)$ must be handled separately. As they correspond to the alignment of a prefix of S_1 to a gap in S_2, $M(i,0)$ can be defined as ig. Likewise, $M(0,j) = jg$. The value in the final cell of the matrix, $M(n,m)$, is by definition the best score for an alignment of $S_{1_{1\ldots n}}$ and $S_{2_{1\ldots m}}$, which corresponds to the best global alignment of S_1 and S_2. The optimal alignment itself can be constructed by following

the path of choices that led to this final value, a procedure known by the name of *traceback*. In this manner, the algorithm finds a single alignment of an optimal score. For more details on this basic approach for global alignment, see (Durbin et al., 1998, chap. 2) and (Rosenberg, 2009, chap. 1).

In the case of equally scoring derivations at a particular point, the classical algorithm above makes an arbitrary choice between them. However, this algorithm can be modified to recover more than one optimal scoring alignment by using a sequence graph structure (Altschul and Erickson, 1986; Hein, 1989). The modified algorithm takes $O(nm)$ time and $O(nm)$ memory, and since n and m are usually comparable, the algorithm is quadratic. A version that uses less memory is given by (Myers and Miller, 1988) and in order to speed up the classical method, *banded dynamic programming* can be used (Chao et al., 1992).

Local alignment solutions. Blast heuristic Above we assumed that we are looking for the best match between two sequences from one end to the other. A more common situation is when looking for the best alignments between substrings of our initial sequences, see Section 1.4. Unexpectedly, the NW algorithm can be easily adapted to local alignment. This local version of dynamic programming alignment algorithm was developed in the early 80s and it is known under the name of Smith & Waterman (SW) algorithm (Smith and Waterman, 1981).

As local alignment is mostly employed in the case of large sequences, in this section we shall give a special attention to the heuristics that are used in this case, mostly due to speed issues. Heuristic approaches sacrifice some sensitivity, with the risk of missing the best scoring local alignments, in order to speed the computation. For this, they introduce a trade-off between two competing parameters: specificity directly affecting the speed of the algorithm and sensitivity affecting its precision, *i.e.* the number of relevant alignments missed. Achieving a good trade-off between sensitivity and specificity is the key issue in local alignment tools. Fasta (Lipman and Pearson, 1985; Pearson and Lipman, 1988), Blast (Altschul et al., 1990) Gapped-Blast (Altschul et al., 1997), *i.e.* Blast2, BlastZ (Schwartz et al., 2000) (an experimental version of the Gapped-Blast program) and PatternHunter (Ma et al., 2002) (based on the optimal spaced seed technology) are examples of such heuristic tools.

Below we chose to detail one of the best-known algorithms, Blast, a state of the art method in the field. In Section 3.6.1 we describe a more recent approach, YASS (Noé and Kucherov, 2005) that uses the recently introduced spaced seed technique, allowing it to increase the sensitivity without loss in specificity.

Blast package provides several programs that search databases with a query sequence (that can either be DNA or protein) and look for high scoring local alignments. The basic idea, common to heuristic methods above, is that true alignments are likely to contain a stretch of identities or a very high scoring match, *i.e.* *seeds*. So, one can start by looking for such seeds that can be further on extended in search for longer alignments. Below we detail the classical Blast strategy. However, the strategy described below allows numerous variations regarding the way hits are extended,

the data structures in which words are kept, the choice of the parameters and the implementation of the different phases.

- As biological sequences have locally biased base compositions, *i.e. regions of low-complexity* (*e.g.* $A + T$ or $G + C$ rich regions), and repeated sequence elements, this may confuse the local alignment programs that will therefore produce matches of little interest. The first step of all different versions of Blast is using filters to eliminate such regions.

- Second, it makes a set of *high-scoring words* of fixed length k scoring more than a given threshold in the query sequence (usually 3 bases for protein sequences and 11 for DNA sequences). Thus, a k-word in the query may be represented by no words in the set or by many. Different tools implement different variations on this approach. *E.g.* Fasta, a precursor of Blast, uses a similar approach but cares for all k-words in the query and not only high-scoring ones. A newer version of Blast (Altschul et al., 1997) searches for two non-overlapping word pairs on the same diagonal, within a limited distance of one another. To achieve comparable sensitivity to the initial Blast algorithm, the threshold parameter for word scores must be lowered, yielding more hits than previously. However, because only a small fraction of these hits is used in the next step, the average amount of required computation decreases.

- Next, it scans through the database, and wherever it finds a word from the previous set, *i.e.* seed, it tries extending it as an ungapped alignment, *i.e. high-scoring segment pairs* (HSPs), in both directions. The extension for one seed does not stop until the accumulated total score of the HSP begins decreasing below a given threshold. As for the scanning phase of the database to be fast, high-scoring words are kept into efficient structures, like search trees. Extending only to ungapped alignments is in fact the technique used in the initial version of Blast. In a more recent version however, seeds are also extended to gapped alignments, see (Altschul et al., 1997).

- Finally, it lists all of the HSPs in the database whose scores are high enough to be considered, meaning that they are greater than an empirically determined cutoff score (by comparing the alignment scores to random sequences).

1.6.2 Multiple alignment

In a multiple alignment, homologous residues among a set of sequences are aligned together in columns. Except for trivial cases of almost identical sequences, it is not possible to unambiguously identify a single correct multiple alignment. Thus our ability to define correct alignments varies with the relatedness between sequences; for cases of interest, where sequences have diverged, one should expect conserved core structural elements that are meaningfully aligned but also other regions whose alignment is not meaningful at all. See Figure 1.5, extracted from (Wallace et al.,

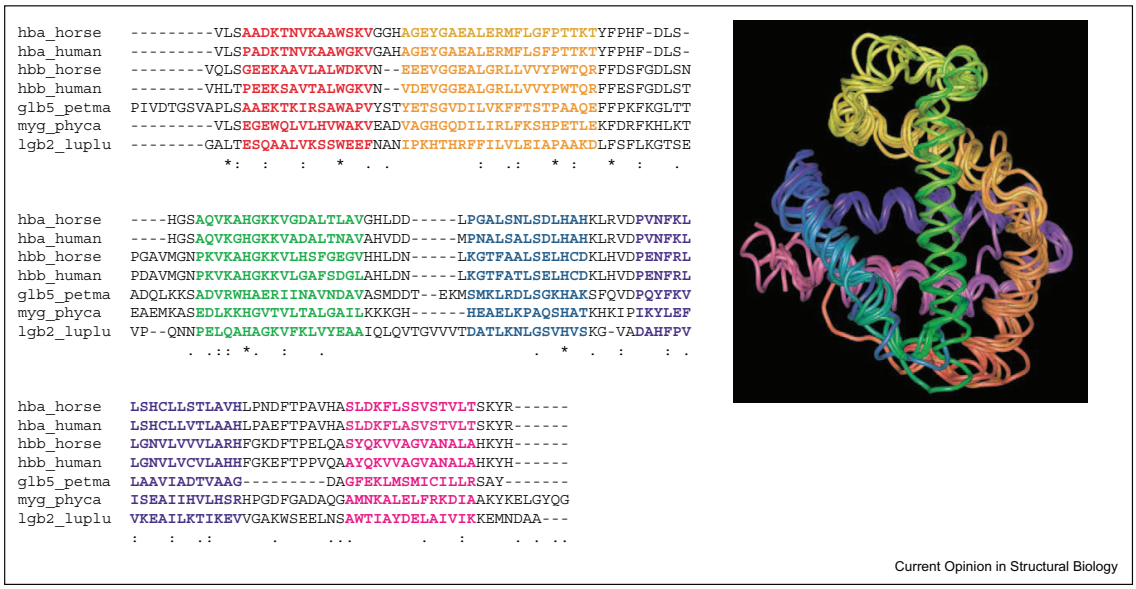


Figure 1.5: ClustalW (Thompson et al., 1994) sequence alignment (left) and VAST structural alignment (right) of seven globins, visualized using Cn3D. The coloured regions within the sequence alignment correspond to the coloured regions within the superposition. Both VAST and Cn3D are available on the NCBI web site http://www.ncbi.nlm.nih.gov/Structure. The figure was extracted from (Wallace et al., 2005).

2005), for an example of a MSA computed for several protein sequences, members of the same protein family, *i.e.* globins, and the corresponding three-dimensional representation.

In theory, one can extend the formulation and algorithm in Section 1.6.1 to more than two sequences, thus for global multiple alignment (*e.g.* by building a three dimensional cubic matrix for three sequences) (Jue et al., 1980; Murata et al., 1985). Unfortunately this would necessitate $O(n^k)$ time and memory, *i.e.* n corresponds to the maximum length among the k sequences. So computing the exact multiple alignment by multidimensional dynamic programming is feasible only for few short sequences. Even with improvements like reducing the volume of the multidimensional matrix as in (Carrillo and Lipman, 1988), one can only use this method for less than 7 sequences of small lengths (< 300 residues).

Therefore, unlike pairwise algorithms, practical multiple alignment algorithms are heuristic rather than exact solutions. By searching only a subset of the population of alignments, they efficiently find an alignment that is approximately optimal but is not guaranteed to be the most optimal alignment possible for the given cost function.

The approach for multiple sequence alignment that really caught on is known as *progressive alignment* (Corpet, 1988; Feng and Doolittle, 1987; Higgins and Sharp, 1988; Thompson et al., 1994; Thompson et al., 1997).

Progressive alignment methods usually start by aligning closely related sequences, as they likely result in a reliable alignment. For this, they build a *guide tree* meaning a quick and dirty tree built with similar techniques to those used to construct phylogenetic trees, *i.e.* the Neighbor-Joining method (Saitou and Nei, 1987). The leaves of a guide tree correspond to the input sequences. These sequences and, upper in the tree, groups of sequences, *i.e.* alignments of alignments, are aligned two by two till a single alignment is obtained. The complete multiple alignment can be found in the root node. Finding a satisfactory guide tree remains however the most time consuming step of this method and recently, several groups worked on improving it (Edgar, 2004b; Edgar, 2004a; Katoh et al., 2002; Katoh et al., 2005). The progressive alignment technique is implemented in widely used programs, like ClustalW (Thompson et al., 1994), Muscle (Edgar, 2004a), T-Coffee (Notredame et al., 2000).

The drawback of progressive methods is that errors introduced in early phases of the procedure cannot be corrected. Several iterative procedures were thus proposed to improve the quality of the alignment (Sankoff et al., 1976; Hogeweg and Hesper, 1984). One common solution based on iterative refinement consists in using hidden Markov models (HMMs).

Hidden Markov models (HMMs) are probabilistic models assigning likelihoods to possible alignments, to determine the most likely multiple sequence alignment or a family of possible alignments. HMMs can produce both global and local alignments and they can be adapted for both pairwise and multiple alignment. For a detailed explanation on how HMMs work, see (Rabiner and Juang, 1986) and (Durbin et al., 1998).

An efficient dynamic programming search procedure adapted to HMMs, *i.e. Viterbi algorithm*, is generally used to successively align the growing MSA to the next sequence in the query set to produce a new MSA, (Hughey and Krogh, 1996; Eddy, 1995; Baldi et al., 1994). Compared to progressive methods, the alignment of prior sequences is updated with each new sequence being added. However, like progressive methods, this technique can be influenced by the order in which the sequences in the query set are integrated into the alignment, especially when the sequences are distantly related. Moreover, HMM optimization can be easily trapped in a local optimum and is unstable in gap-rich regions. Finally HMMs do not explicitly consider evolutionary relationships among sequences.

Among other multiple alignment programs mostly for small-scale sequences we cite 3D-Coffee (O'Sullivan et al., 2004) for protein sequence alignments, MAFFT (Katoh et al., 2002; Katoh et al., 2005) that ingeniously uses a fast Fourier transform in order to speed the progressive alignment strategy, PROBCONS (Do et al., 2005), very close to T-Coffee but more accurate thanks to the use of pair-hidden Markov models and MUSCLE (Edgar, 2004b; Edgar, 2004a), an extremely fast and accurate method based on *log* expectation.

For a thorough description of multiple alignment algorithms and applications see (Gotoh, 1999) and (Higgins and Taylor, 2000, chap. 3). Additional information on recent improvements on multiple alignment methods and short descriptions of such novel methods can be found in (Wallace et al., 2005; Notredame, 2007).

1.7 Benchmarks, simulations and comparison of alignments

Face to the growing number of alignment applications, a vast array of alignment programs have been developed. Moreover, in the last ten years, thanks to the new high-throughput genomic and proteomic technologies, the size and complexity of the data sets that need to be routinely analysed are increasing. Ideally, when new methods are developed it is necessary to evaluate the improvement compared to existing methods. Moreover, a detailed evaluation should be performed to determine the performance of the different methods under different conditions, thus allowing to choose the most appropriate one for a given alignment problem. This should also guide future research on alignment algorithms.

1.7.1 Alignment benchmarks

Benchmarking is widely used in computer science to compare the performance of different technologies. Within a scientific discipline, it captures the community consensus on which problems are worthy of study, and determines the scientifically acceptable solutions. The characteristics that benchmarks should have in order to be considered successful, *i.e.* leading to a faster progress and a more rigorous examination of research results, are mentioned in (Sim et al., 2003). They must be: *relevant*, *i.e.* include tests that need to be representative of the problems the system must handle, *solvable*, *i.e.* achievable but not trivial tasks with solutions known in advance, *accessible*, *i.e.* tests and results need to be publicly available, and *evolving*, *i.e.* the benchmark needs to continuously develop in order to keep the pace with the optimization rhythm.

In the field of sequence alignment, when building a benchmark three main issues need to be explored. First, what is the *correct* alignment of the sequences included in the tests? Second, which particular alignment problems should be represented in the benchmark? Finally, measures for comparing to the reference alignment need to be specified. Moreover, in the particular case of sequence alignment, benchmarks need to quickly evolve in order to provide larger test cases, more representative of the new alignment requirements, *i.e.* large scale sequence alignment.

(McClure et al., 1994) is one of the first studies to compare the quality of different alignment methods. It uses four sets of sequences from four different protein families and shows that the performance of alignment programs mainly depends on the number of sequences, the degree of similarity between sequences and the number of insertions in the alignment. Moreover, factors as the length of the sequences, the existence of large insertions and the over-representation of some members of the protein family,

may also affect alignment quality. In the same study, McClure *et al.* measured the ability of existing methods to identify conserved motifs and concluded that in general global methods performed better than local methods. The relevance of these tests was however limited, due to the small size of the benchmark. Since then, several pairwise and multiple sequence alignment benchmarks were created for protein sequences: Homstrad (Mizuguchi et al., 1998), BAliBASE (Thompson et al., 1999b), OXBench (Raghava et al., 2003), Prefab (Edgar, 2004b), SABmark (Van Walle et al., 2004). See (Blackshields et al., 2006) for a comparison of the main benchmarks for protein sequence alignment algorithms and (Aniba et al., 2010) for a survey on this subject.

For RNA sequences, BRAliBASE (Gardner et al., 2005) is the only benchmark designed to evaluate and compare alignment programs. For DNA sequences on the other hand, there are no adapted benchmarks and therefore there is a real shortage of assessments of MSAs with real DNA data. One solution to this problem, in the case of coding DNA sequences, would be to compare computed nucleotide alignments against reference nucleotide alignments that are based on the biological features used in protein benchmarks. Little work has been done to address this lack of reference DNA alignments. Recently, (Carroll et al., 2007) introduce the first known collection of protein-coding DNA benchmark alignments based on biological features such as the tertiary structure of encoded proteins. These solutions are unfortunately of limited use, as they only deal with protein coding DNA, while nowadays the interest of scientists turns to non-coding DNA, the so-called *junk* DNA, see Section 1.1.

In (Pollard et al., 2004) a benchmarking tool for the alignment of non-protein coding DNA was created, using simulated data. Though this benchmark gives researchers a starting point to evaluate DNA alignments, the degree to which the simulated sequences reflect those in nature is uncertain.

1.7.2 Simulations

Simulations are designed in order to model reality. In its simplest form, the evolution of a DNA sequence can be modelled through a Markov process. It starts with an initial sequence and emulates the mutation process by randomly choosing sites within the sequence and allowing them to mutate to another nucleotide with a probability based on a theoretical model of nucleotide change, such as the Jukes-Cantor model (Jukes and Cantor, 1969), the Kimura two-parameter model (Kimura, 1980), or the HKY model (Hasegawa et al., 1985), among others. Protein sequence evolution can be simulated in the same manner by using amino acid change matrices such as Dayhoff (Dayhoff et al., 1978) or JTT (Jones et al., 1992).

More complex experiments simulate sets of sequences across a phylogenetic tree. In the simplest case, branching rates in the tree are assumed to be equal. Involved models include different rates of evolution at different sites, different models of evolution for different segments of sequence (separate modelling of functional domains, exons or introns), or different models of evolution across different parts of a tree.

Simulations have a number of undeniable advantages. First, we can know the exact and precise truth about the data. For the alignment, we know exactly which specific

sites in a sequence are homologous to sites in the other sequences. We know with absolute certainty the phylogenetic relationships and the history of the sequences. In fact, for empirical data sets we never know the absolute truth and we can only infer it from the data thus, in most cases, truths are merely hypotheses. Second, by completely controlling the simulations, we can direct the tests on the effects of specific variables and conditions, by leaving constant every aspect of the simulation but the one we want to test. However, the main inconvenient of simulations is realism, as our understanding of mutational processes is incomplete and the simulation can only be as good as the model of the evolutionary process that underlies it.

The first widely used software for simulating alignments was Rose (Stoye et al., 1998), which simulates a family of related sequences (RNA-, DNA-, or protein-like) guided by a known evolutionary tree. It is one of the first simulation programs including indels. With the help of their program, the authors showed that the optimal alignment is not necessarily the correct one. Lately, additional sequence simulation programs have been developed, like Simali (Blanchette et al., 2008) based on Rose, Dawg (Cartwright, 2005) and MySSP (Rosenberg, 2005b) for DNA sequences, Indel-Seq-Gen (Strope et al., 2007) and EvolveAGene3 (Hall, 2008) for DNA and protein sequences.

1.7.3 Comparing alignments

One issue that needs to be discussed is how one should actually compare alternate alignments, eventually with some form of benchmark alignment.

When comparing alignments from a program to some *benchmark alignment*, two measures are usually used in order to assess the similarity between them: *the sum-of-pairs score* (SP) and *the column score* introduced in (Thompson et al., 1999a). The *SP score* is defined as the percentage of correctly aligned pairs of residues in the alignment produced by the program, compared to the reference alignment. It is used to determine the extent to which programs succeed in aligning some of the sequences, if not all of them. The *column score* corresponds to the percentage of correctly aligned columns in the alignment. It tests the ability of the programs to align all of the sequences correctly.

Concerning gaps, neither one of the two measures takes them into account in the computation. They deal with them as follows: the SP score ignores paired comparisons containing gaps, while the column score computes the measure based only on residues within the column. Moreover, both measures above consider only correctly aligned residues. In OXBench benchmark (Raghava et al., 2003), an alternative approach was included, *i.e. the Position Shift Error score*, which measures the average magnitude of error by penalizing shifts between alignments. All of these scores are usually computed in the regions of the alignment identified as being reliable, *i.e. the alignment backbone*, see Section 2.1.3.

Among these measures, the most appropriate one depends on the set of sequences to be aligned and the requirements of the user. For example, in the case of divergent sequences, the *SP score* may be more useful since the programs will not be able to

align all sequences correctly; the *Position Shift Error* score can also be useful in this case. The *column score* is more meaningful for alignments containing many closely related sequences and only a small number of divergent sequences. As most alignment programs correctly align closely related sequences, this generates a high *SP score*, even if the divergent sequences are misaligned. In this case, the *column score* discriminates better among programs correctly aligning the divergent sequences compared to the others.

When comparing several potential alignments, thus without a reference alignment, the measures are sensibly similar to those above. The classical solution in this case, is considering one by one each alignment as a reference alignment and compute scores relatively to this reference. An average between the scores computed in this way is then given as a final score. Such approaches can be found in (Lassmann and Sonnhammer, 2002; Rosenberg, 2005a).

Measures described above compare alignments at the pair of residues level or at the column level. Another approach would be to use global measures, which compute global scores for each alignment, and then compare these scores. Such three fundamental types of scores exist in the literature. The first type of scores refers to the *coverage* and the *percentage identity*. One may find different definitions for these measures but the most common are as follows. The *percentage identity* is computed as the number of identical pairs of residues in the alignment on either the length of the alignment or the lengths of the sequences. The *coverage* corresponds to the total length of the regions covered by the alignment. For more details regarding these measures see Section 3.3.

The second type of score is a *raw Smith & Waterman score*, see Section 1.6, meaning the measure optimized by the SW algorithm. It is computed by summing the substitution matrix scores for each position in the alignment and subtracting gap penalties. As we have seen in Section 1.6, it is difficult to discriminate between such scores. To overcome this problem, a third type has been introduced, the *statistical score* based on the extreme value distribution (EVD). The power of statistical scores can be attributed to their incorporation of more information than any other measure; it takes into account the full substitution and gap data (like raw scores) but it also incorporates details about the sequence lengths and composition (Brenner et al., 1998).

2 State-of-the-art. Whole genome alignment

More than 1000 complete genomes are available for bacteria and archea, and another 6000 genomes are ongoing. Because of this vast amount of data, comparative genomics analyses at whole genome scale become mandatory. Moreover, as most probably only few reference genomes will be consistently annotated in the next future, a need for robust and rapid genome based comparison tools not relying on gene annotations has emerged. Several advances have been recently achieved in terms of complete genome comparison tools not relying on gene annotations. Due to genome sizes and the task complexity, such tools generally implement heuristic methods, *i.e. the anchor-based strategy*. With the flood of data and tools, there is also an increasing urge for improved datasets and protocols for evaluating the correctness and performance of genomic alignment software.

Contents

2.1 Introduction to the alignment of whole genomes

A complete genome sequence of an organism may be considered to be the ultimate genetic map, in the sense that the heritable characteristics are encoded within the DNA and that the order of all the nucleotides along each chromosome is known (Hardison, 2003). One gains much insight into genome structure by analyzing such individual genome sequences, however in order to understand genome function one sole sequence is not enough material (Venter et al., 2001; Lander et al., 2001). In fact, a present challenge for genome research is to distinguish functional DNA and assign roles to it (Collins et al., 2003). Recently, genetic maps have been created for a wide range of species and more recently even, DNA sequences of whole genomes have been sequenced in a great number. This has led to the field of research called *comparative genomics*, which analyzes characteristics of whole genomes, in contrast to *comparative genetics* that analyzes single genes only.

The statement of comparing genomes covers a multitude of types of studies: studies that focus on *homologous genes* (two genes that are both related by descent from a gene in a common ancestral species) and look for how many homologous genes they share and their order, or that are interested in comparing the protein structures and functions, or that completely leave out genes and look at intergenic entities such as *transposable elements*. Probably one of the most important applications of comparative genomics is the functional annotation of genomes. Functional parts of sequences are subject to negative selection, which causes slower changes than in the case of nonfunctional sequences. In principle, by comparing genomic sequences one can find signatures of selection and infer that the corresponding sequence parts are functional. Moreover, the role of such sequences can also be predicted. As both analysis are predictions, experimental tests must be done, like those being currently implemented on a large scale in the *Encyclopedia of DNA Elements* (ENCODE) project (Weinstock, 2007; Ewan Birney et al., 2007). For a thorough introduction to biological, statistical and computational questions in comparative genomics see (Sankoff and Nadeau, 2000) and (Balding et al., 2003, chap. *Comparative genomics*).

There are three main kinds of approaches used for whole genome comparisons: approaches based on the *genome as a "set of genes"*, *whole proteome comparison* (Tatusov et al., 1997) and *whole genome sequence alignment* studies. All three approaches are powerful tools to study genome organization and evolution rules with different time scale considerations. For example, the information contained in the order in which genes occur on the genomes of different species was used successfully for reconstructing phylogenetic relationships, see (Gascuel, 2005, chap. 12) for a review. The last two types of approaches have been employed with success in a recent study comparing the genome of yeast *S. Cerevisiae* to three related yeast species genomes. This comparative analysis of the yeast chromosomes has considerably improved gene annotation and has permitted the prediction of new motifs conserved in intergenic regions that act potentially as regulatory elements of gene expression (Kellis et al., 2003; Kellis et al., 2004).

2.1.1 The need for whole genome alignment (WGA) at DNA level

In particular, whole genome comparisons based on alignments at the DNA-level started to develop very recently. The first sequenced genomes were evolutionary distant and DNA alignment tools were not sufficiently powerful, therefore comparisons were limited to protein sequences. In fact, exons of protein coding genes mutate substantially slower than the non-coding DNA, as mutations changing an amino-acid are less likely to be retained in a functional protein-coding gene (Li and Olmstead, 1997). Based on comparisons at the amino-acid level, a large set of common proteins were identified in yeast, worms and flies (Rubin et al., 2000), but non-coding sequences could not be compared at that point. Genome alignments at the DNA-level were first used for the comparison of whole genomes on species that are more closely related. An example of such study on strains of bacteria from several species: *Escherichia coli*, species of *Salmonella*, and *Klebsiella pneumonia* can be found in (McClelland et al., 2000).

Advances in the whole genome alignment (WGA) field became urging with the recent developments in the area of comparative genomics. The most striking change in this area is the sheer volume of available data due to the novel sequencing techniques (Miller, 2001). At `http://www.genomesonline.org/cgi-bin/GOLD/bin/gold.cgi`, one learns that 1364 genomic sequences have been completely published at the moment, among which 92 archaeal, 1139 bacterial and 133 eukaryal. Moreover, another 190 archaeal, 4897 bacterial and 1581 eukaryotic genomes are ongoing and to be published in the near future.

If one says great amount of data from, till now, little unknown species, one implies little knowledge and poor annotation of sequences. In situations like these, approaches working at the level of the proteome become less helpful. Additionally, an evolution of the research subjects has happened in the field, as researchers started to give an important interest to non-coding regions (already discussed in this section and in Section 1.1). Nowadays, one is interested in identifying conserved sequences in both coding and non-coding regions. This made the use of WGA tools at the DNA level mandatory.

Five years ago, such a quantity of data at such speed seemed out of the question. This defines novel demands for DNA alignment tools, which should be capable of dealing with both a great number of sequences and huge lengths. Frequently, WGA tools have to work with incomplete, "draft", sequence data consisting of pieces whose relative order and orientation are difficult to determine. Unfortunately, little work has been done on the subject. Moreover, comparison of complete genomes implies genome shuffling events. Mechanisms like repeated inversions or translocations determine a reordering of genetic elements (Tillier and Collins, 2000). In the case of bacteria, horizontal transfer introduces new genetic elements (Hacker and Carniel, 2001). Furthermore, the apparition rates and patterns of such events depend on the set of genomes being compared. For example, one is more likely to observe gene duplications and repetitive sequences in eukaryotes than in bacteria (Eichler and Sankoff, 2003). Genome comparison systems must account for all of these mechanisms in order to provide a complete picture of genetic differences among organisms.

2.1.2 The reality of whole genome aligners

Whole genome alignments are still in an exploratory phase, even though a great number of dedicated tools were created during the last years. See (Miller et al., 2004; Blanchette, 2007; Miller, 2001; Treangen and Messeguer, 2006) for reviews on algorithmic ideas and on available resources for computing, accessing, and visualizing genomic alignments.

As classical alignment methods based on the dynamic programming strategy, see Section 1.6 cannot be adapted to the case of long, divergent and shuffled genomic sequence, WGA tools are generally heuristics based on a complex strategy composed of several phases, (Brudno et al., 2003b; Hohl et al., 2002; Delcher et al., 1999; Darling et al., 2004; Treangen and Messeguer, 2006). This strategy, known under the name of *anchored-based strategy*, is meant to combine speed and sensitivity for genomic alignment. In a first step, a fast search tool is used to identify sequence similarities. Then, these similarities are filtered by using a chaining phase and the selected ones are used as anchor points for the final alignment. Finally, a more sensitive method aligns those regions that are left unaligned between the identified anchor points. See Section 2.2 for additional details.

In (Miller et al., 2004) the following needs for WGA tools have been identified:

1. improved software for aligning two genomic sequences on rigorous statistical basis;
2. reliable and automatic software for aligning three or more genomic sequences;
3. improved datasets and protocols for evaluating the correctness and performance of genomic alignment software;
4. better methods for displaying and browsing genomic sequence alignments.

The points enumerated above remains of great relevance nowadays, regardless the great amount of work having been done in this domain.

The context of this manuscript does not extend to the problem of visualizing and browsing WGA, even though this is an extremely challenging problem in itself. In this chapter and in Chapter 3, we shall focus on the first three points raised by W. Miller. Below, we summarize the actual state of WGA methods and identify some of their limitations and future directions of improvement. In Chapter 3, we give ideas meant to improve WGA by addressing several of these limitations, and experimental results sustaining our claim of improvement.

Algorithmic choices The strategies used by the WGA methods (see Section 2.2) are composed of several phases and for each of these phases, WGA tools implement different algorithms. The choice of these algorithms directly influences the speed and the quality of the final alignments. We shall see in Section 2.4 and in Section 2.5 that most tools use, at every step of the strategy, variations on the same methods. However,

no study has been dedicated to measuring at which point these variations influence the final result. Moreover, as such algorithmic solutions are similar and as they follow similar strategies, the question that one should ask oneself is at which point they are interchangeable, meaning that they could be plugged in with each other in order to create new methods. Such ideas, consequences and underlying results are discussed in Chapter 3.

Parameters Many WGA programs have a great number of parameters for each one of the phases composing the strategy, parameters that may greatly affect the output, are poorly understood and difficult to be adjusted. A very recent work on accurate parameters for local alignments when used on complete genomic sequences (employed as hits in the first step of WGA methods) can be found in (Frith et al., 2010). For this study, 495 combinations of score parameters for alignment of animal, plant, and fungal genomes were assessed, and it was found that the scoring schemes used in the UCSC genome database are far from optimal, respectively the X-drop parameter and the E-values. However, except for parameters used in the first phase of the strategy, parameters in the other phases, seldom common between different WGA tools, also need to be fixed, *e.g.* thresholds for the sizes of matches, fragment weights, maximal admitted distances between fragments when clustering them, number of recursive phases. Authors usually make their choice on empirical criteria that are more or less documented. To our knowledge, there is a lack of detailed studies regarding these choices and their impact. Some of these parameters are also discussed in Chapter 3.

Quality of alignments and comparison of different WGA solutions Differences in behaviour of WGA tools are difficult to predict and results even more difficult to analyze as there are no statistical or empirical criteria available to evaluate the quality of genome alignments. With few established methodology for estimating the quality of the WGA, scientists have either trusted the alignments completely or developed their own filters. Not being capable of estimating the quality of alignments complicates the task of comparing and evaluating objectively the results of different WGA tools. See Section 2.6 and Chapter 3 for a more detailed discussion on these subjects.

2.1.3 Applications of whole genome alignment at DNA level

Regardless these persisting problems, whole genome alignment tools have lately been used in a variety of studies: for the identification of regulatory sites (Chapman et al., 2003; Dubchak et al., 2000; Göttgens et al., 2000), prediction of genes in eukaryotes (Alexandersson et al., 2003; Bafna and Huson, 2000; Carter and Durbin, 2006; Flicek et al., 2003; Korf et al., 2001; Stanke et al., 2006), identification of signature sequences for pathogenic microorganisms (Chain et al., 2003; Fitch et al., 2002), detection of functional noncoding RNAs (Pedersen et al., 2006; Washietl et al., 2005), motif identification (Bigot et al., 2005; Halpern et al., 2007), phylogenomic studies (Delsuc et al., 2005).

We already discussed that due to large structural changes, *i.e.* rearrangements, the organization of genomic sequences is often modified. Regions that have been affected by rearrangements are called *breakpoints*, while the ones that have not been touched by such events are referred to as *synteny* blocks. Precise knowledge on breakpoint regions should lead to a better understanding of the course of evolution and its functional impact (Claire and Sagot, 2008). In order to study breakpoint related questions, one has to precisely identify the genomic regions that underwent rearrangements. This is usually done with WGA methods allowing for rearrangements (see Mauve, CHAINNET, Shuffle LAGAN described in Section 2.5); breakpoint regions identified in this manner are then refined and narrowed (Lemaitre et al., 2008), and thoroughly studied as to investigate issues related to hotspots (Bourque et al., 2005) or to the correlation between fragile sites, segmental duplications, and rearrangement locations (Bailey et al., 2004; Armengol et al., 2003).

One easily observes that the vast majority of genomes already sequenced or close to being completed, are bacterial genomes. In fact, since the publication of a second strand of *Helicobacter pylori* in 1999 (Alm et al., 1999), sequence data on related bacterial genomes has rapidly accumulated in public databases. Thus, a considerable interest is given to the study of bacteria, as the availability of sequences for multiple strains of multiple species opens up new perspectives for studying short term evolutionary processes (Chiapello et al., 2005).

One among the first comparative studies for bacteria was based on the pairwise alignment of complete genomes of *Escherichia coli* (Sakai vs K12) and allowed the identification of a highly conserved subsequence between the two strains (Hayashi et al., 2001; Perna et al., 2001). They found that this common subsequence is in fact the conserved part of chromosome, among the strains of *E. coli*.

Mosaic structure of bacterial genomes The conserved part among different strains is termed the *backbone*. Is is interrupted by DNA segments specific to each strain, called *variable segments*. This type of organization of genomic sequences in backbone and variable segments is known as the *mosaic structure of genomes* (Chiapello et al., 2005; Chiapello et al., 2008).

Studying the mosaic structure of bacterial species is extremely relevant as the backbone is more likely to be related to the essential functional elements of the cell (genes, motifs or signals) (Hayashi et al., 2001; Perna et al., 2001; Lefébure and Stanhope, 2009), while variable segments most probably correspond to mobile elements (Schneider et al., 2002; Vishnoi et al., 2007) and with high probability they might be putative locations for the discovery of pathogenic islands.

Obtaining backbone/variable segments segmentation at a large scale, *i.e.* for a high number of species each of them with several strains, needs competitive automatic tools for whole genome alignment at the DNA level. Initial studies were limited to pairwise comparisons and only recently systematic strategies for such segmentations have been proposed in (Chiapello et al., 2005; Chiapello et al., 2008). The latter work was used to analyze the mechanisms of genetic variability in *E. coli*, *S. aureus*, and *S. pyogenes*,

and to analyze recombination, based on an alignment of backbones obtained from a comparison of 20 *E. coli*/*Shigella* strains. In Chapter 3 we shall discuss the quality of this systematic analysis of backbone/variable segments segmentations with existing WGA tools.

2.2 Strategy for WGA

From the mid-1980s, with the arrival of the personal computer, hundreds of sequence alignment programs and algorithms have been published. Moreover, from the mid-1990s, tools for aligning extremely long sequences have appeared and rapidly reached dozens. In Figure 2.1 extracted from (Treangen and Messeguer, 2006), one has an approximate, and by no means complete, overview of alignment tools, including small-scale sequence alignment and whole genome alignment tools. WGA tools need to be able to efficiently handle comparisons involving mega-bases of genomic sequence. Unfortunately, traditional alignment methods would require days and even months of computation time even on competitive computers. Thus alignment tools are adapted to the sizes of the sequences they deal with, *i.e.* small sequences, considered as such if they are less than $10Kbp$, or large ones, more than $10Kbp$.

Small-scale sequence alignment tools and methods have been described in the previous chapter. Here, we focus on WGA tools, and more precisely on the strategy upon which such tools are usually based. By looking in the phylogenetic tree in Figure 2.1, one distinguishes rapidly several possible classifications of WGA tools, based on the number of sequences they align, *i.e.* pairwise and multiple, based on the level of shuffling of the sequences they deal with, *i.e.* tools designed for collinear sequences (without large scale rearrangement events) and tools capable of handling rearranged sequences.

Among **pairwise WGA solutions** one may identify a unifying thread: the organization in four phases known as the *anchor based strategy*, which can be summarized as follows. For a graphical representation of the phases composing the anchor based strategy, see Figure 2.2.

1. **Preliminary phase.** Before the first phase, most WGA tools use a preliminary filtering step (or demand to the user to apply this kind of filtering) that consists of masking low-complexity DNA regions, *i.e.* simple repeats, namely micro-satellites (Turnpenny and Ellard, 2005) and $A+T$ or $G+C$ rich regions, and interspersed repeats (Hess et al., 1983). This preliminary filtering is meant to avoid uninformative and time-consuming alignments among repeats.

2. **Fragments computation phase** This first phase consists in the detection of exact matching regions between the sequences, or in the case of more recent tools, detection of approximate matches, *i.e.* allowing a small number of mismatches and gaps. Such matching regions are known under the names of *fragments* or *potential anchors* and they work as seeds for the following of the method.

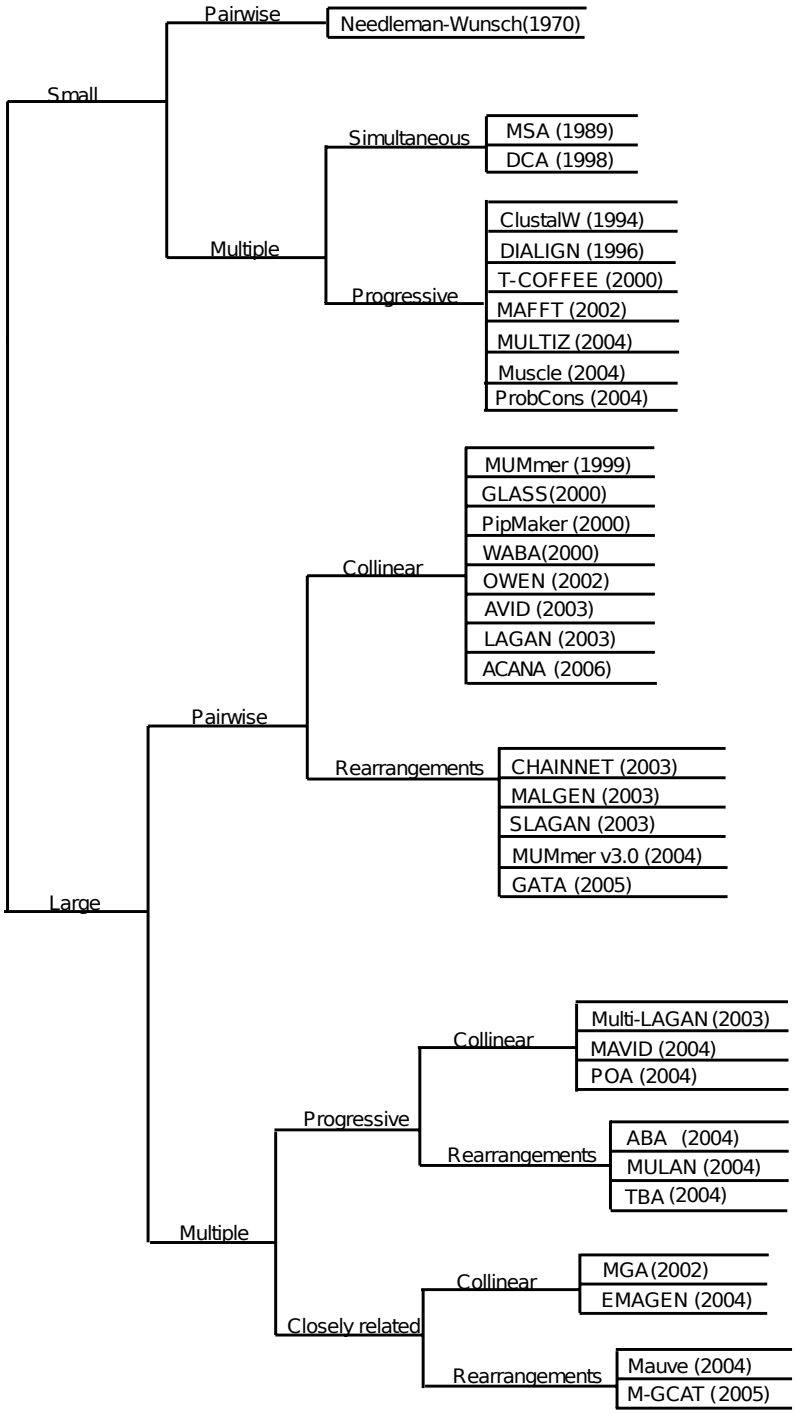


Figure 2.1: "**An approximate phylogeny of genome comparison tools over the past 30 years.** Tracing the growth in related global genome comparison tools over the past 30 years." Figure extracted from (Treangen and Messeguer, 2006).

To be able to efficiently deal with entire genomes, the idea behind this *seeding* phase is to limit the dynamic programming search space. In fact, methods for computing different types of fragments are fast, *i.e.* they run in linear time, and they allow an initial identification of interesting regions, more likely to be part of true homologous subsequences. However, in order to avoid a number of spurious matches, fragments are usually filtered by imposing a minimum allowable size, adapted to the lengths of the genomic sequences, or in the case of approximate matches, a minimum level of quality, *e.g.* a percentage identity cutoff. This filtering step prevents from generating a great number of fragments that could be due to random similarity between sequences, thus facilitating the task for the second phase. Details on the methods that are employed for computing fragments can be found in Section 2.3.

This first phase is probably the most important one in the strategy, as the quality of the alignment mostly depends on the sensitivity of this phase. Using exact or approximate matches of varying lengths and with varying characteristics may greatly influence the final result. Thus working on this phase in order to improve the quality of the WGA methods becomes mandatory. This point shall be discussed in Chapter 3.

3. **Chaining phase** The second phase corresponds to the selection of a subset of fragments. This step is an additional filtering of matches in order to fix the boundaries of the final alignment. The fragment selection can be made on different criteria depending on the characteristics of the sequences the tool is supposed to align, *i.e.* collinear or shuffled sequences. This step is usually known under the names of *chaining* or *anchoring* and the fragments that are selected at this step are called *anchors*, as they form the basis of the final alignment and cannot be discarded in the following phases.

   The most common chaining strategy is intended for collinear sequences, meaning that it looks for an *optimal collinear sequence of non-overlapping fragments*, *i.e.* a set of increasing fragments on both sequences. Several exact solutions exist for this variation of the chaining problem and they are thoroughly discussed in Chapter 4.

   For shuffled sequences, the restriction of collinearity of fragments has to be relaxed in order to accommodate eventual rearrangement events. An intermediary solution is to impose the order between fragments on one sequence only, meaning that on the second sequence, fragments may be taken in any order, see Section 4.11. This however, is not the perfect answer to our problem, as fragments on the second sequence may overlap. The complete solution is for one to look for an *optimal set of non-overlapping fragments*, meaning that no constraint is put on the order of the fragments in the two sequences, see Section 4.10 for a complete definition. In this case, tools implement heuristics, like the *greedy breakpoint elimination technique* described in 4.12, or the hierarchical greedy strategy used in (Roytberg et al., 2002) and (Morgenstern et al., 1998).

4. **Recursive phase** The first and the second phases described above are usually applied in a recursive manner on *gaps* in between the anchors, *i.e.* yet unaligned pairs of regions. This corresponds to a *divide-and-conquer* strategy. The goal is to examine the genomes for as much matching genomic sequence as possible by searching the regions that lie between anchors for additional shorter matches, thus creating new regions small enough to be efficiently aligned in the last phase. Filters that are used in the first step to avoid spurious matches are being relaxed when the phase is repeated, *i.e.* adapted to sequences that are considerably shorter than the initial ones. This allows for a more sensitive research of similarities, capable of digging out more divergent homologous regions.

5. **Last chance alignment phase** As the name suggests it, this fourth phase consists of systematically applying small-scale sequence alignment methods on all remaining unaligned regions, *i.e.* gaps, usually below a certain length threshold.

> **Aligned gaps** We refer to these remaining unaligned regions, aligned by the *last chance alignment phase*, as *aligned gaps*, a term that was first introduced in (Darling et al., 2004).

Both pairwise and multiple WGA tools are mostly based on the anchor strategy. In the particular case of multiple WGA, methods for computing fragments capable of dealing with more than two sequences are needed. As we shall see in the second part of the manuscript, chaining methods for the multi-dimensional case exist. The third and the fourth phases are implemented on the same principle as for the pairwise case. In the fourth phase, meant to close the gaps between the anchors, the programs ClustalW (Thompson et al., 1994) and Muscle (Edgar, 2004b) are currently employed (see Section 1.6.2 for a description of these two programs). For example, Mauve uses the initial matches in order to compute a distance between each two sequences, on which it bases further on the construction of a guide tree with the *Neighbor Joining* method (Saitou and Nei, 1987). In this way, it avoids computing a new guide tree for each two subsequences that it aligns with ClustalW.

In the following of the chapter, we detail the first phase of the anchor strategy and some of the most common tools implementing it. We shall no longer insist on the *third* and the *fourth phase*, as they tend to have little variation from one tool to another and as there is little description on them.In Section 2.3 we summarize the basic techniques employed in the *first phase*, *i.e.* the fragments computation phase. The *second phase* algorithms are thoroughly explained in chapters 4 and 5. In Section 2.4 we describe the most commonly used among tools designed for pairwise WGA, while in Section 2.5 the ones dealing with multiple alignment.

2.3 First phase of the anchor based strategy

The *first phase* of the anchor based strategy consists in identifying similarity regions among the sequences, *i.e.* *fragments*, regions that represent potential anchors for the

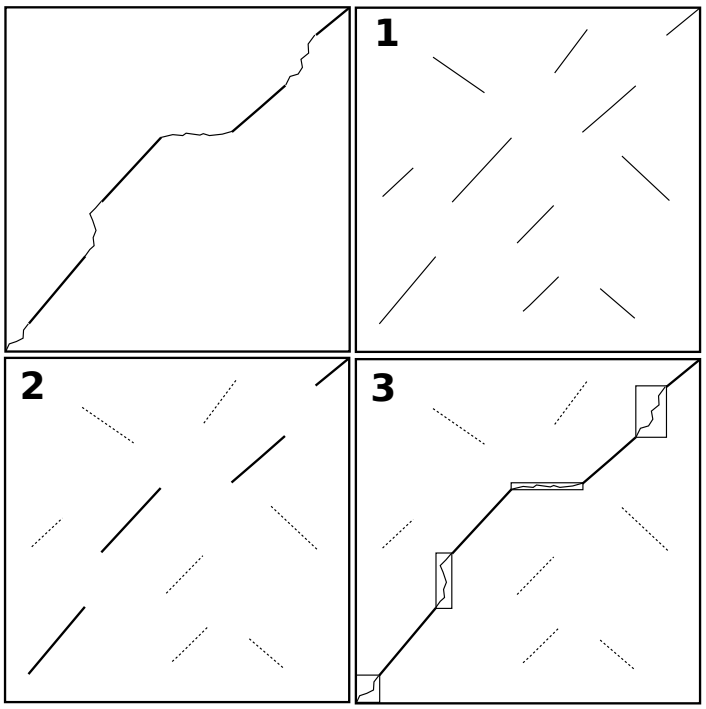


Figure 2.2: The anchor based strategy. The first plot corresponds to a pairwise collinear WGA, *i.e.* a path between the bottom left and the top right corner of the similarity matrix, the closest possible to the main diagonal. (1) *Fragments computation phase*. Segments in the plot correspond to fragments, *i.e.* matching regions between the two sequences. (2) *Chaining phase*. In this example the chain consists of a collinear set of anchors; there is also the version allowing for rearrangements in the chain. (3) *Recursive anchoring* in gaps between the anchors and finally, alignment of still not aligned regions.

final alignment. Thus the quality of the final WGA is essentially determined by the quality of this initial set of similar regions.

2.3.1 Types of fragments

Fragments usually employed in the first phase of anchor based methods for WGA are *exact matches*. An elementary type of exact matches initially used by WGA tools, consists of *exact k-mers*, *i.e.* pairs of identically matching substrings of length k. Once identifying k-mers, one usually extends them to maximal-length matches, see tools like sim2 (Chao et al., 1995) and sim3 (Chao et al., 1997); this kind of approach is similar to the Blast-like hashing scheme described in Section 1.6.1. More recent approaches are based on the direct identification of *maximal unique matches (MUMs)*, see (Delcher et al., 1999).

Maximal unique matches (MUMs) More precisely, a MUM is a sequence occurring exactly once in each genome, which is not contained in any longer such sequence, see Figure 2.3. The motivation for using this kind of match is that if a long, perfectly matching sequence occurs exactly once in each genome, it is almost certain that it should be part of the alignment.

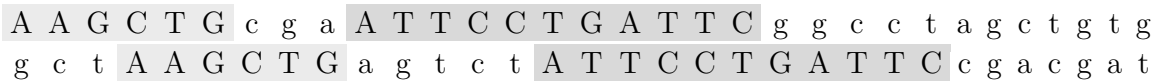


Figure 2.3: Two *maximal unique matching sequences* (MUMs) shown in uppercase, shared by the two sequences. Any extension of the MUMs will result in a mismatch. Any contraction of the MUMs will result in a non maximal match. By definition, a MUM does not occur anywhere else in either genome, thus a sequence like *cga* is a match, but it is not unique.

Variations on exact matches Except for computational reasons, the restriction of using only MUMs, *i.e.* maximal **unique** matches, as anchors seems unnecessary and it is not justified from a biological point of view. Exact matches occurring more than once in a genome may also be meaningful.

Thus, using *maximal exact matches (MEMs)* was the following innovation after the use of MUMs. MEMs are more general than MUMs as they are allowed to have several occurrences on each genome. Similar to a MUM, a MEM is defined as a substring that occurs in the genomes to be aligned and cannot be simultaneously extended to the left or right in every genome. It is however true that by relaxing the constraint of uniqueness, one has to be more efficient in the chaining-filtering phase in order to choose meaningful anchors of the alignment. New matches meant to combine the advantages of both MUMs and MEMs, are *maximal rare matches (MRMs)*, meaning exact matches that occur at most k times on each sequence, where k is usually a small number.

If fragments based on k-mers, MUMs and MEMs are the ones that are the most commonly used, variations on these can be found in the literature. For example, WABA (Kent and Zahler, 2000) treats *wobble bases*, *i.e.* third residues in codons, differently from other bases, as it is known that mutations at this residue are often silent in the sense that they do not change the corresponding amino-acid. The strategy of WABA, is similar to that of Blast, only that for WABA the HSPs are not required to match exactly but may contain a mismatch every three residues.

Approximate matches In order to accommodate sequences with higher degrees of divergence, matches with degeneracies, *i.e. approximate matches*, were also employed in the literature. Such approximate matches are for example k-mers matching with at most c differences between the sequences, or ignoring translations and transversions, see (Brudno et al., 2003a), the method used in the first phase of LAGAN and Shuffle

LAGAN. Moreover, recent tools like Mauve (Darling et al., 2004) and its successor ProgressiveMauve (Darling et al., 2010), use approximate matches based on *spaced seeds*, see Section 3.6.1 for a description of spaced seeds.

A recent tool that assembles the types of matches that can be used in the first phase of the anchor based strategy is VMatch (Kurtz, 2003), an extension of the older REPuter (Kurtz et al., 2001). It efficiently solves large scale multiple sequence matching tasks and can compute different kinds of matches, using state-of-the-art algorithms, like: maximal exact matches, maximal unique matches, maximal repeats, branching tandem repeats, supermaximal repeats and complete matches. Matches can be selected according to their length, E-value or score. VMatch can also realize several post-processings of the initial matches, as extending or clustering them.

2.3.2 Algorithmic solutions for computing exact matches

Unlike sim2 and sim3 that use hash-like methods to compute exact matches, see Section 1.6.1, most recent methods are based on *suffix-trees* (Gusfield, 1997, chap. 5), which directly find maximal matches that may be unique or not.

Suffix trees The name of *suffix tree* refers to a compact representation storing all possible *suffixes* of an input sequence S, *i.e.* a suffix is a substring beginning at any position in the sequence, which extends to the end of it. The suffix tree is a rooted directed tree with as many leaves as characters in S. Each internal node, except the root, has at least two children and each edge is labelled with a non-empty substring of S. Edges out of a node have labels starting with different characters. The defining property of a suffix tree is that for any leaf i, the concatenation of labels on the path from the root to the leaf i spells out the suffix starting at position i. See Figure 2.4 for an example of suffix tree built for a given sequence.

Note that each internal node in the suffix tree corresponds to a repeat in S, with the repeat number being equal to the number of leaf nodes below that node in the tree. Based on this observation, one may use suffix trees for computing matches between two genomic sequences, as following. First, the two genomic sequences, S_1 and S_2, are concatenated in order to obtain a single sequence, *i.e.* $S = S_1 \& S_2$ where & is a symbol neither occurring in S_1 nor in S_2. Second, a suffix tree is built for the newly obtained sequence S, like in (Delcher et al., 1999). As it is straightforward that a MUM is represented by an internal node with exactly two child nodes, such that the child nodes are leaf nodes from different genomes, MUMs can be easily identified.

Querying a suffix tree to identify MUMs can be done in linear time and space. And, if the basic algorithm to construct suffix trees is quadratic in the size of the sequence, several solutions for building suffix trees in linear time exist, *i.e.* (McCreight, 1976; Ukkonen, 1995; Weiner, 2008). Moreover, by using *enhanced suffix arrays* (Abouelhoda et al., 2004), which have an index structure that requires much less space than the classical implementation (Kasai et al., 2001), solutions become even more efficient, see VMatch (Kurtz, 2003).

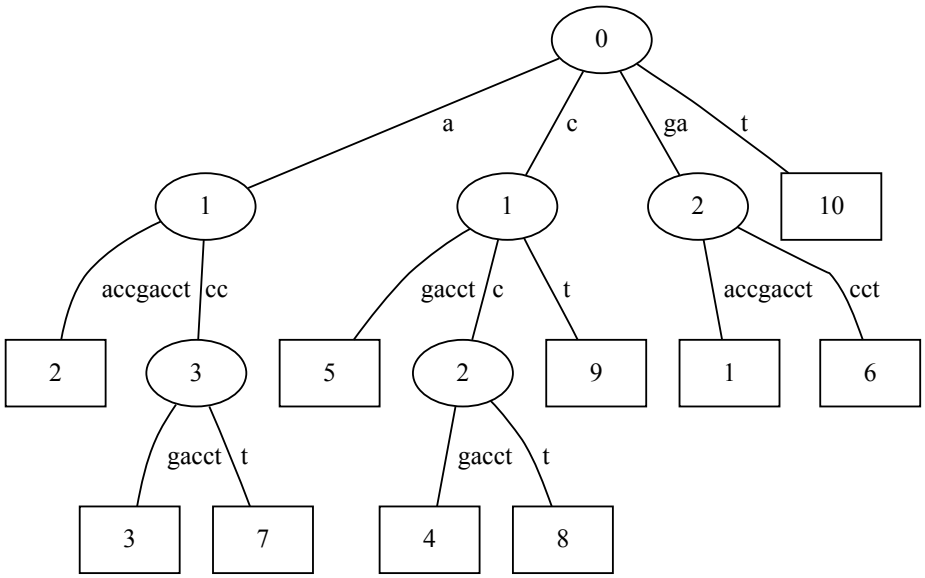


Figure 2.4: Suffix tree for the sequence *gaaccgacct*. Square nodes are leaves and represent complete suffixes. They are labelled by the starting position of the suffix. Circular nodes represent repeated sequences and are labelled by the length of these sequences. In this example the longest repeated sequence is *acc*, occurring at positions 3 and 7. Figure adapted after (Delcher et al., 1999).

Conclusion As detailed in the current section, fragment computation methods employed in the first phase of the anchor based strategy compute short, exact/approximate matches. However, one may want to use *local similarities* instead of such matches, which seem likely to be better suited to the comparison of whole genomes, especially when dealing with divergent sequences. This idea has been recently employed in ProgressiveMauve (Darling et al., 2010), a WGA method described in Section 2.5.2 based on *ungapped local similarities*. See chapters 3 and 6 for a study on the impact of local similarities in WGA methods and for a novel WGA method based on local similarities.

2.4 Pairwise WGA tools

In this section we detail the main tools designed for pairwise WGA, all of them being variations on the *anchor based strategy*. Based on Figure 2.1 where pairwise alignment tools are classified on whether they deal or not with rearrangements, we organise the current section as follows: first, we describe in Section 2.4.1 the tools designed for collinear pairs of genomes, while in Section 2.4.2 we address methods dealing with rearrangements.

2.4.1 Pairwise WGA for collinear sequences

MUMmer (Delcher et al., 1999)

The arrival of MUMmer produced a major breakthrough in the alignment of whole genomes. MUMmer is based on three phases that can be shortly described as follows:

1. **Fragments computation phase** This first phase provides a MUM decomposition for the two genomes. The minimal admitted length of a MUM is empirically fixed, given the level of similarity of the genomes. For highly similar genomes for example, their default length threshold is set to $50bp$. For more distant sequences, this parameter is usually adjusted to $20bp$, as it is about exact matches.

2. **Chaining phase** In order to select anchors, the initial matches are filtered with a collinear chaining method allowing for overlaps (see Section 4.3). The authors refer to a simple quadratic algorithm as a potential solution for this chaining problem, but only a short description of it is provided. It seems that overlaps between matches are allowed in the chaining algorithm and removed further on by shortening matches. As a result of this post-processing, they obtain a suboptimal chain, with no insurance on the length and quality of the chosen matches. Even though they mention the the algorithm in (Jacobson and Vo, 1992) as a more efficient alternative solution, they do not test it in practice.

3. **Last chance alignment phase** Gaps in between anchors less than or equal to a given limit ($5000bp$ by default) are closed with the NW algorithm, described in Section 1.4. Gaps longer than this limit remain unaligned.

Observe that MUMmer does not apply recursively the first and the second phases, thus losing in sensitivity compared to other WGA methods. The idea of using only two phases instead of four, is however interesting and must be looked into further on, see Chapter 3. Even though revolutionary for its time, MUMmer had the disadvantage to deal with only two, very close, genomic sequences and to be slow and memory demanding. An improved variation of MUMmer appeared three years later and it was described in (Delcher et al., 2002) as being three times faster and needing three times less memory than the original version. Same as for VMatch (Kurtz, 2003), this improvement is based on the use of enhanced suffix arrays (Abouelhoda et al., 2004).

GLASS (Batzoglou et al., 2000)

GLASS appeared not long after MUMmer. It however implements a slightly different version of the anchor based strategy, as it was designed for serving a particular purpose: to be the pre-processing step of a gene prediction tool, ROSETTA. It is thus conceived based on the established model of eukaryotic DNA sequences, *i.e.* containing long, weakly conserved introns and short, strongly conserved exons, thus not really adapted to prokaryotic sequences.

1. **Fragments computation phase** It searches for all exact matching k-mers, initially $k = 20$. For each one of the k-mers, GLASS applies a dynamic programming procedure to 12 residues at the left of the k-mer and to 12 residues at its right. Based on this procedure, it computes a score for each k-mer.

2. **Chaining phase** In order to compute a collinear chain of k-mers, the two genomic sequences are converted into strings of characters, as following: one character replaces a k-mer, each occurrence of a k-mer in the sequence is replaced by its corresponding character. k-mers only are represented in these strings, basically corresponding to exon parts, while the weakly conserved introns are completely ignored. Next, k-mers are filtered with a simple dynamic programming procedure computing the maximal scoring collinear chain. Moreover, k-mers taken in the chain are additionally filtered if they have scores below a certain threshold and if they are inconsistent, *i.e.* their overlap on one sequence is different from the overlap on the second sequence. The resulting set of matches corresponds to the anchors of the alignment.

3. **Recursive phase** The first two steps are recursively applied on gaps between anchors with decreasing lengths for the k-mers, *i.e.* 15, 12, 9, 8, 7, 6 and 5.

4. **Last chance alignment phase** An intermediary step before the final phase consists in extending the anchors by short local alignments in both directions. Finally, all remaining gaps are aligned with a standard dynamic programming algorithm.

In (Hohl et al., 2002), it has been argued that the major drawback of GLASS is its huge space requirement and running time. Further more, as GLASS was entirely thought for eukaryotic sequences, *i.e.* dealing with fixed length k-mers that are only extended in the last phase, it cannot take advantage from long identical regions in the two sequences, which are very common in prokaryotes. Experiments confirming this hypothesis were conducted on the initial $200Kbp$ of two strains of *E. coli*, for which GLASS was stopped after 25 hours of computation time.

The previous programs were both intended for the alignment of pairs of collinear genomic sequences. Besides these two, plenty of solutions exist, some of which we cite below with few details:

- WABA (Kent and Zahler, 2000), whose first key feature was already described in the previous section, *i.e.* taking apart wobble bases from the others. After a Blast-like phase, WABA aligns regions around HSPs using pairwise hidden Markov models (Durbin et al., 1998). The clear disadvantage of this method is being impractical for large genomes.

- OWEN (Roytberg et al., 2002) proposes an interesting approach: their solution is based on the observation that in case of conflict between several similarities (rearrangement events or overlaps), one should keep a strong similarity with

respect to its score, instead of seeking to optimize a global scoring function. For this, they proceed to a hierarchical greedy method that deals with fragments in descending order of their score and builds a collinear chain that they assume closer to the evolutionary true chain reflecting orthology. Unfortunately, even though the idea seems pertinent, no biological proof was brought to sustain this claim.

- PipMaker (Schwartz et al., 2000), a well known web-based tool that uses BlastZ to compute local alignments between the two sequences, see Section 1.6.1. It implements a chaining procedure, which returns the best scoring set of fragments in the same order on both sequences but no exact description is given of how this is done. The alignment tool is limited to these two phases of the anchor based strategy. PipMaker was tested on the human-mouse comparison, two nematode species and two bacteria, and the results have been compared to the exons predicted by GenScan (Burge and Karlin, 1997). In (Schwartz et al., 2003), the speed of BlastZ was improved, in order to align entire genomes. This was obtained by adding a dynamic masking step and seeding with approximate matches. In the same study, BlastZ was successfully used to verify the conservation of synteny for the human chromosome 20 compared to the mouse chromosome 2.

- AVID (Bray et al., 2003), very similar to MGA, is restricted to pairwise comparisons; it uses MEMs, collinear chaining and recursive alignment. Worthy to mention is that AVID defines an interesting protocol for comparing WGA results of different tools, see Section 2.6.

- LAGAN (Brudno et al., 2003b), whose strategy is similar to its version for rearranged sequences Shuffle LAGAN, described below.

- ACANA (Huang et al., 2005) comes with an interesting idea: it begins by computing local alignments with a heuristic version of the SW algorithm. Based on their scores, it recursively selects anchors in a given region in order to form a collinear chain and finally computes the optimal global alignment for regions between fixed anchors. ACANA is however little known and used in practice.

2.4.2 Pairwise WGA allowing for rearrangements

Most of the WGA tools that deal with rearrangements are also dealing with multiple sequences, as we shall see in the next section. Here we describe some of the few tools dedicated to pairwise WGA in the presence of rearrangements.

CHAINNET (Kent et al., 2003)

CHAINNET proposes one of the first automated solutions for linking together traditional local alignments into several chains in order to accommodate inversions, translocations, duplications and deletions. It starts by computing local alignments with

BlastZ, from which only gapless parts are extracted. Next, three types of chaining strategies are proposed.

1. A variation of the algorithm given in (Zhang et al., 1994), and shortly described in Section 4.9, for building multiple collinear chains. A particular scoring scheme is used, with greater penalties for mismatches than for gaps, meant to advantage the opening of gaps rather than forcing alignments between non-homologous regions. The disadvantage of this approach is that several chains may overlap the same region, in the absence of any restriction.

2. Among chains computed with the method above, the best scoring chain can be selected, *i.e.* it results in a collinear WGA of the two sequences as in the previous section.

3. The third approach, and the most interesting one, builds a type of chain of local alignments, not necessarily collinear, *i.e.* net of chains. The solution consists of marking initially all bases in the chromosomes as unread, then processing chains (obtained with the first chaining method) in descending order of their scores. Parts of chains that intersect with bases already covered by previously taken chains are thrown out. For keeping track of areas of the chromosome that are already covered, the program uses red-black trees. Chains that fill in holes of previous chains are marked as children of these previous chains. The idea, slightly resembling that of (Roytberg et al., 2002), is rather interesting as it allows for rearrangements.

Shuffle LAGAN (Brudno et al., 2003c)

Shuffle LAGAN, a generalization of LAGAN for rearranged genomes, is a classical anchor based approach. It is considered as being the first method for WGA in the presence of rearrangements.

1. **Fragments computation phase** Finds fragments using CHAOS tool (Brudno et al., 2003a), a method that looks for small matching words, *i.e.* k-mers with little degeneracy, which are further on linked together provided they are at a maximum admitted distance and shift from one another. Finally, chains are extended using a Blast-like extension method.

2. **Chaining phase** Compared to LAGAN, which builds a classical collinear chain of fragments, in Shuffle LAGAN a novel type of chain is computed: a *glocal chain* of fragments. This chain is composed of several collinear subchains, as described in Section 4.11.

3. **Last chance alignment phase** The collinear subchains from the previous phase are finally aligned with LAGAN.

Comparisons with Mauve (see Section 2.5.1) on simulated data showed that Shuffle LAGAN works well on divergent data without a big number of repetitive subsequences. The difficulty that Shuffle LAGAN has in dealing with repetitive sequences can be partly overcome by masking such repetitive elements with the program RepeatMasker (`http://www.repeatmasker.org/`), as the authors of the paper propose. In (Darling et al., 2004) it was discussed that this good performance on more divergent genomes appears to be due in part to the simulation method and it is not likely to be observed on real data. However, it is certain that CHAOS, with its local alignments, plays an important role in the sensitivity of Shuffle LAGAN.

Besides CHAINNET and Shuffle LAGAN, the newest version of MUMmer, MUMmer3.0 (Kurtz et al., 2004) also deals with rearrangements, it is fast and uses little memory compared to the initial version. MUMmer3.0 improves on the first version of MUMmer by allowing all maximal exact matches and not only MUMs. Moreover, this time, exact matches are clustered and extended (an idea that we also found in CHAOS). This is done with Nucmer (for nucleotide alignments) and Promer (for amino-acid alignments) programs, which generate local alignments. The disadvantage is that it produces a set of local alignments and not a complete WGA.

2.5 Multiple WGA tools

2.5.1 Tools for closely related sequences

MGA (Hohl et al., 2002)

MGA is considered to be the first tool capable of aligning three or more genomes, though restricted to collinear ones, and it remains nowadays a reference in the domain of WGA. MGA is what one could call a classical implementation of the anchor strategy:

1. **Fragments computation phase** Based on a virtual suffix tree, MGA detects all MEMs in the genomic sequences, above a given threshold. Later, this first phase was implemented separately in VMatch program, capable of doing a large number of different match type searches besides the classical MUMs and MEMs, see Section 2.3.

2. **Chaining phase** Anchors are selected by building a chain of collinear fragments with the classical quadratic algorithm described in Section 4.4. In a more recent tool, *i.e.* Chainer (Abouelhoda and Ohlebusch, 2005), this second phase was developed independently based on an improved algorithm, see Section 4.9.

3. **Recursive phase** MGA closes the gaps between the anchors by recursively applying the first two phases a certain number of times, while lowering the length threshold for the MEMs.

4. **Last chance alignment phase** Remaining gaps, if lower than a given threshold, are aligned with ClustalW. There is also the option of using a percentage identity

threshold: force the alignment of gaps having a percentage identity above a fixed value. Long gaps or low quality gaps remain unaligned as to cope with long insertions/deletions.

Being the first multiple WGA tool, MGA was initially compared to the pairwise aligner MUMmer. Moreover, as the comparisons were designed to assess the memory and the speed of the two programs, MGA was made to reproduce MUMmer strategy, by computing MUMs instead of MEMs. Thus, the real improvement brought by the use of MEMs was not discussed in the paper.

A couple of years after MGA, EMAGEN was published. Compared to MGA, EMAGEN (Deogun et al., 2004) searches for MUMs with the novel technology employed by VMatch, described in (Abouelhoda et al., 2004). Same as MGA, EMAGEN chains fragments in a collinear chain. However the algorithm they use for chaining is similar to that described in Section 4.6, thus faster than the one proposed in the initial MGA publication. While gaps below a certain threshold are closed with ClustalW, EMAGEN has no recursive phase. Thus, though EMAGEN is faster than MGA and with very good results on prokaryotic genomes thanks to the use of the *functional matching* strategy, *i.e.* ignoring non-coding regions when aligning sequences, in practice MGA remains a better alternative for the alignment of collinear genomes due to its increased sensitivity.

Mauve (Darling et al., 2004)

Among the first WGA tools providing global alignments for more than two sequences in the presence of rearrangements, Mauve combines the analysis of large-scale evolutionary events with traditional multiple sequence alignment. Therefore, it adapts the anchor based strategy as follows:

1. **Fragments computation phase** The published version of Mauve works with MUMs, in order to avoid the problems arising with repetitive sequences. Contrarily to MGA, which computes matches directly on the complete set of genomes, Mauve uses a seed and extend method. It starts by finding matches in a subset of genomes only and then extends them to the remaining subset. However, since the published version, Mauve improved its approach by using approximate instead of exact matches, based on palindromic spaced seeds described in (Darling et al., 2006), see Section 3.6.1 for details on spaced seeds. Fragments found in this phase are exploited in order to compute a distance metric used to construct a phylogenetic guide tree based on the *Neighbor Joining* method. The utility of this tree becomes clear in the last phase.

2. **Chaining phase** The second phase consists of a particular heuristic strategy for chaining fragments while allowing for rearrangements. The result corresponds more to clusters than to chains but we chose to remain consistent in the terms we employ, thus we refer to them as chains. Based on the breakpoint analysis technique (Sankoff and Blanchette, 1998), it computes the minimum partitioning

of the set of MUMs into collinear subchains, called Longest Collinear Blocks (LCBs). LCBs correspond to homology regions without rearrangements. A detailed description of the algorithm can be found in Section 4.12; a graphical representation is given in Figure 4.13.

3. **Recursive phase** As the initial anchoring step may not be sensitive enough to detect the full regions of homology within and surrounding the LCBs, several steps of recursive anchoring are applied. Unlike other genome aligners performing a fixed number of recursive passes, Mauve computes a minimum anchor size based on the lengths of the respective subsequences and stops recursive anchoring when either no additional anchors are found or when the region is shorter than a fixed length. Thus, regions **in between LCBs** are first concatenated and then the new search for MUMs is done with parameters adapted to shorter sequences. Regions that were not unique in the initial configuration may become unique outside the LCBs, therefore new MUMs may be discovered. Pairs of facing unaligned regions **inside LCBs** are searched separately for determining novel matches.

4. **Last chance alignment phase** Facing regions between anchors inside LCBs are aligned using the progressive alignment method, ClustalW, based on the guide tree computed in the first phase. Mauve refers to these unaligned facing regions as *aligned gaps* (see page 41, Section 2.2). Unaligned regions above $10kb$ are left unaligned and they correspond to specific regions, horizontal transfers, or paralogous repetitive regions that may be identified as such during processing with other tools.

Mauve was implemented as a genome alignment package with a visualization environment that depicts the rearrangement events. Mauve was thoroughly tested and compared to other tools on simulated data. Given a rooted phylogenetic tree, the sequences were obtained with a nucleotide substitution model implemented in the Monte Carlo simulation package called Seqgen (Rambaut and Grass, 1997); small insertions and deletions were modelled as occurring with uniform frequency and distribution throughout the genomes. Moreover, their model included horizontal transfer events and inversions. Translocations, even though not explicitly taken into account, can result from two inversion events. A dataset containing nine sequences resembling to nine enterobacteria was generated in this manner. The experiments consisted of comparisons with Multi-LAGAN (Brudno et al., 2003b), the extension of LAGAN to the multiple alignment case (briefly described in Section 2.5.2), and Shuffle LAGAN, at the moment the only genome aligner available, capable of dealing with rearrangements (for Shuffle LAGAN only two sequences were considered for comparisons).

As already explained for Shuffle LAGAN, the fact that both Multi-LAGAN and Shuffle LAGAN use anchors containing substitutions and indels, makes them more sensitive thus more appropriate for sequences with a high level of nucleotide substitutions; for lower levels of substitutions, Mauve obtains similar results to Multi-LAGAN or Shuffle LAGAN. This problem is partly solved in ProgressiveMauve (see

Section 2.5.2), with the use of approximate matches as anchors. Compared to Shuffle LAGAN, Mauve clearly excels at aligning rearranged sequences. In conclusion, Mauve seems to be better suited to closely related sequences that underwent modest amounts of nucleotide substitutions. The experiments in Section 3.5 brought us to the same conclusion, meaning that Mauve is very well adapted to sequences having a low level of divergence but it has a hard time in dealing with divergent sequences.

M-GCAT (Treangen and Messeguer, 2006)

M-GCAT is another multiple genome alignment and visualization system, a close relative of Mauve, published shortly after. Indeed, it was designed to align large, multiple, closely related genomes involving rearrangements. M-GCAT searches for MUMs common to all genomes, based on a recursive approach. First an initial set of matches is found, and then, a set of decreasing parameters is used in order to identify shorter MUMs between the established anchors. Spurious matches are filtered with respect to a minimum length parameter and then grouped into clusters containing collinear matches within a maximal admitted distance (similar to the LCBs of Mauve). M-GCAT was compared to Mauve regarding their ability to identify orthologous genes, *i.e.* the percentage of known orthologous genes, as specified by COG (Tatusov et al., 2000), located in the same clusters. These tests reveal Mauve's increased accuracy compared to M-GCAT, probably due to the use of approximate matches, but differences are however small between the two methods. Moreover, compared to Mauve, M-GCAT is faster and deals with larger sets of genomes.

2.5.2 Progressive alignment methods

The four tools previously described are designed for closely related genomes. In theory, for comparing more distant genomes, one uses progressive alignment tools, instead those built on the anchor based strategy. Progressive methods like Multi-LAGAN (Brudno et al., 2003b) and MAVID (Bray and Pachter, 2003) for collinear genomes and ProgressiveMauve (Darling et al., 2010), an improved version of Mauve for rearranged genomes, build an alignment of k genomes in $(k-1)$ pairwise alignment steps, based on a phylogenetic guide tree. The guide tree can be random and iteratively refined, or it can be considered as being known. The major shortcoming of this kind of approach is that it can introduce a bias in the inference of the ordered series of homologous regions, due to a possible over-representation of a subset of sequences.

ProgressiveMauve (Darling et al., 2010)

Very recently, a novel version of the Mauve algorithm, ProgressiveMauve, was published in (Darling et al., 2010). This novel algorithm addresses some of the limitations of the original one. The original Mauve algorithm aligned regions that are conserved among *all* organisms. Unfortunately, the portion of a genome conserved among all genomes shrinks as more sequences are added to the analysis. Therefore, as it could

not align regions conserved among a subset of the genomes under study, the original Mauve algorithm did not scale well to large numbers of genomes. ProgressiveMauve strategy can be described as follows:

1. **Fragments computation phase** Instead of using a single seed pattern for match filtration as in the latest versions of Mauve, ProgressiveMauve uses a combination of three palindromic seed patterns for improved sensitivity, see (Darling et al., 2006). Seed matches, which represent a unique subsequence shared by two or more input genomes are subjected to ungapped extension until the seed pattern no longer matches. The result is an *ungapped local multiple similarity* with at most one component from each of the input genome sequences.

2. **Computing a guide tree** ProgressiveMauve uses a guide tree in order to progressively align the genomes. For building the guide tree, it employs the *Neighbor Joining* method based on the shared gene content among each pair of input genomes.

3. **Anchor selection based on progressive genome alignment** A genome alignment is progressively built up according to the guide tree. At each step of the progressive genome alignment, anchors are selected from the initial set of local multiple alignments as to maximize a sum-of-pairs scoring scheme.

4. **Global alignment** Once anchors have been computed at a node in the guide tree, they serve to build a global alignment. Given a set of anchors among two genomes, an alignment or a pair of alignments, ProgressiveMauve applies a modified Muscle global alignment algorithm to compute an anchored profile-profile alignment.

5. **Filtering the alignment of unrelated sequence** Although we compute a global alignment among sequences, genomes often contain lineage-specific sequence that makes them globally unrelated. Therefore global alignment will seldom generate force alignment of unrelated sequence. A simple hidden Markov model structure is used to detect forced alignment of unrelated sequence, which are then removed from the alignment.

Compared to Mauve, ProgressiveMauve can be applied to a much larger number of genomes, it can align more divergent genomes, it usually does not need manual adjustment of the alignment scoring parameters and it aligns regions conserved among subsets of the input genomes. Moreover, thanks to the filtering phase, ProgressiveMauve obtains highly accurate results, as observed in Section 6.3 and Section 6.5. However, it is substantially slower than the original Mauve algorithm and it consumes more memory than the original Mauve algorithm.

Multi-LAGAN & MAVID for collinear sequences

Multi-LAGAN needs a phylogenetic tree as input and uses LAGAN to align the closest two sequences in the tree; it obtains what it calls a multi-sequence. Among the remaining sequences, two by two the closest ones are aligned. Next, it mounts in the tree and aligns two by two the closest multi-sequences.

MAVID starts with a random binary guide tree for the set of genomes and it recursively aligns the sequences at ancestral nodes with AVID program. At each node of the tree, ancestral sequences are inferred from the existing alignments using maximum likelihood. Once the multiple alignment is completed, a novel phylogenetic tree is inferred with the *Neighbor Joining* method. Based on the novel guide tree, the procedure for obtaining a multiple alignment is repeated. The whole process is performed three times altogether, *i.e.* tree + alignment. Judging by the experiments published (Bray and Pachter, 2003), MAVID seems to be more accurate and faster than Multi-LAGAN.

MABA & TBA for distant sequences

As we have seen in Section 1.6.2, Figure 1.5, a multiple alignment is generally represented as a linear sequence of alignment columns. This implies a global alignment of sequences that are similar over their entire length. It is precisely the assumption that tools like Multi-LAGAN and MAVID make. An interesting discussion on the relevance of a linear representation of MSA can be found in (Lee et al., 2002). In answer to this debate, a representation replacing the classical one was proposed, based on a directed acyclic graph instead of a single directed path as in the classical linear column representation. This kind of representation can depict additional evolutionary events like recombinations of similarity regions, *e.g.* very common in the evolution of multi-domain proteins (Doolittle, 1995). Still this type of representation is not flexible enough to capture the entire complexity of biological sequence comparisons, as regions of similarity may be shuffled or repeated. Below we describe two methods that further improve the MSA representation.

For ABA (Raphael et al., 2004) an alignment representation with high-multiplicity edges, allowing directed cycles, was proposed. Their graph representation, based on A-Bruijn graphs, is perfectly suited to the alignment of protein sequences composed of several domains that may be: not present in all proteins, present in different orders in different proteins, present in multiple copies in the same protein. Moreover, ABA is adapted to the alignment of genomic sequences rich in duplications and inversions. They were the first to adapt A-Bruijn graphs to MSA, *i.e.* an A-Bruijn graph is an extension of the classical de Bruijn graph first described in (De Bruijn, 1969; Good, 1946). de Bruijn graphs have already been used in bioinformatics problems (Pevzner et al., 2001; Li and Waterman, 2003; Zhang and Waterman, 2003), while A-Bruijn graphs were only recently introduced and applied to fragment assembly and repeat

identification (Pevzner et al., 2004). Starting with a collection of pairwise similarities between each two sequences, ABA computes a graph corresponding to a collection of paths, where edges represent rearrangements of similarity regions.

TBA (Blanchette et al., 2004) is another multiple alignment tool, published right before ABA, which represents a multiple alignment as a set of *alignment blocks*, *i.e.* local alignments involving some (or all) of the sequences that contain every position of each sequence only once. This set of blocks is called a *threaded blockset*. Even though the concept of threaded blockset easily accommodates rearrangement events, the version of TBA that we describe below works under the assumption that there are no inversions, duplications or other rearrangement events.

The particularity of TBA is that it does not fix a reference sequence for its blockset. The drawback when using a reference sequence is that regions conserved in a subset of sequences but not in the reference sequence, are inevitably missed. Moreover, alignments obtained with different reference sequences, may differ and may even be inconsistent, *i.e.* positions aligned to each other in one situation, are aligned to different positions in another situation. No matter how TBA blocks are threaded, *i.e.* by choosing a reference sequence and projecting blocks on it, one always obtains a consistent set of blocks. Due to their consistency, one could obtain a classical global alignment by simply sticking blocks together.

BlastZ (Schwartz et al., 2003) is used to compute the pairwise local alignments between the original sequences, and MULTIZ (Blanchette et al., 2004) to compute alignments between three or more sequences. Threaded blocksets are built as following: sequences are added progressively to the blockset along a phylogenetic tree, from leaves to the root. The blocks in the blockset at a node (different from the leaves) result from intersections of the blocks at its children nodes, based on a guiding pairwise blockset. The blocks are only split into smaller blocks during the progressive steps of TBA, they are never expanded. The problem with such method is choosing the reference sequences when running the pairwise alignments in order to build the guiding pairwise blockset. Nothing guarantees that the alignments of these reference sequences can be extended by transitivity to the other sequences in the blocksets. Moreover, splitting blocks without closely analyzing the newly obtained subsequences and their alignments, can be tricky, as one is not sure of the quality of such blocks.

TBA threaded blocksets are similar to the representation with high-multiplicity edges introduced by ABA. In fact, ABA can be used to automatically generate blocksets for TBA, even though the two algorithmic approaches are very different. The published version of TBA however, deals only with similarity regions occurring in the same order and direction in the sequences, while leaving open the problem of identifying in an automatic manner blocksets resulting from inversions, duplications and other rearrangement events. Thus, this representation has the same drawbacks as the one using directed acyclic graphs. As it removes restrictions on order, direction and repetitions, ABA's approach represents a solution to the open problem left by TBA. Note that since its published version, TBA can now handle inversions.

TBA (Blanchette et al., 2004) and ABA (Raphael et al., 2004) approaches are both revolutionary solutions that produce (or could produce) multiple alignments of sequences including shuffled and repeated regions, a feature that lacks in other alignment methods. However, they do not give a precise alignment at residue level, in order to obtain a complete multiple alignment, such tools need to be combined with classical alignment methods. First they are used to uncover the structure of the multiple alignment and then, similarity regions discovered in this manner have their alignments refined with classical, progressive multiple alignment methods like ClustalW, Muscle.

Other solutions

Even though progressive multiple alignment seems to be the best answer to MSA of distantly related sequences, alignment solutions not relying on a phylogenetic tree exist.

One among such solutions is called *consistency-based MSA* and it was first introduced in the DIALIGN programs (Morgenstern et al., 1998). It starts, similar to tools described above, by building pairwise local alignments. This time, pairwise anchors are scored not only on the similarity of the two subsequences involved in the local alignments, but also on the similarity of the other pairwise anchors aligning these two subsequences in the other genomes. Thus a pairwise anchor with low similarity between the two subsequences may find its score augmented thanks to a third subsequence in a third genome that aligns well with both of them. Next, optimal pairwise alignments are built. Anchors that compose them are sorted according to their scores and their overlaps on the other sequences (in order to emphasize motifs occurring in several sequences). Finally, the multiple alignment is computed in a greedy manner. The particularity of this method is that the greedy step needs to keep the "consistency of the alignment", *i.e.* without conflicting assignment of residues among the set of sequences. The greedy approach was replaced by a probabilistic method in ProbCons (Do et al., 2005) designed for protein sequences and Pecan (Paten et al., 2008) for genomic sequences, presently used by the *Ensembl browser*.

2.6 Estimating quality and comparing WGA

Sequence alignment and in particular WGA are extremely difficult computational problems. Researchers rely on the results of the alignment tools and are usually accustomed to viewing their favourite genome aligned against other genomes, and examine their zones of interest in the alignments. This is however imprecise and extremely fastidious. Moreover, one has no means to know what the true solution is and without a correct alignment as reference, a precise evaluation of accuracy cannot be done. Thus, there is presently a great need for methods to assess whole-genome sequence alignments and compare the alignments produced by different tools. Such methods should be able to determine which tool is most accurate, especially on distantly related

species, whether there is a difference in quality between the alignments of coding and non-coding regions, whether methods that align a bigger percentage of the sequences give accurate results on the additional aligned regions.

In order to assess the quality of the alignments produced by WGA tools, simulated datasets are usually used (Darling et al., 2004; Blanchette et al., 2004; Treangen and Messeguer, 2006). However, the quality of the evaluation in these cases is as high as the simulation of the evolutionary history of the genomic sequences is faithful. In this manuscript, we do not insist on this aspect, given that we chose to evaluate our work on real data, instead of simulated sequences, see Section 3.2. Below we describe approaches designed for the evaluation of alignments on real data.

2.6.1 Accuracy on regions with known features

When examining protein coding regions, one can benefit from the extensive results on the subject, see Section 1.7.1. For example LAGAN and Multi-LAGAN were both tested on protein-coding exons in orthologous sequences collected from the following datasets: ROSETTA set containing 129 orthologous annotated genes between human and mouse (Batzoglou et al., 2000) and the CFTR region consisting of 12 orthologous sequences from human and another 11 species (Thomas et al., 2003a). The evaluation done in (Brudno et al., 2003b), measures the capacity of LAGAN and Multi-LAGAN to correctly align pairs of orthologous genes. However, one should be careful to avoid over-fitting of tools for homologies in coding regions, as this could generate bad results in non-coding regions. Moreover, orthologous sets are usually identified using some aligner thus implying a certain circularity in the process. Finally due to the limited size of the gene sequences (compared to a complete genomic sequence), these results do not inform us on their ability to align whole genomes.

The problem becomes more difficult for non-coding regions, since the right-answer is not known. Even though one may extract examples from databases containing experimentally confirmed regulatory sites like (Kolchanov et al., 2002) and estimate the quality of the alignments in these regions (Stojanovic et al., 1999; Wasserman et al., 2000), this can only be of little help because these kinds of regions represent only a small proportion of the non-coding regions.

With this type of approach, one has no information on the accuracy in regions in-between known features. Moreover, in this way, tools that are especially designed for WGA and whose interest is to correctly distinguish orthologous regions from the others, are evaluated at the nucleotide level on short, relatively similar regions. In fact, a good evaluation in this case says nothing on the real quality of these tools when aligning whole genomic sequences.

2.6.2 Finding suspicious regions in alignments

Without orienting the analysis of alignments on known regions, one wants to be able to verify the alignment as a whole and to distinguish suspicious from certified quality regions. For this, basic filters can be applied on the pieces of the alignments in order to

keep only high quality ones, *e.g.* a cutoff on the percentage identity or on the score like in (Chiapello et al., 2005; Chiapello et al., 2008; Bray et al., 2003). However, cutoffs are generally difficult to set, given that they should depend on multiple factors, *i.e.* the level of divergence of the sequences, whether the piece of alignment corresponds to a coding region or not.

StatSigMA-w method, assessment of homology regions An involved method for examining precisely the quality of whole genome multiple alignment can be found in (Prakash and Tompa, 2007). They classify alignment regions into well aligned and suspiciously aligned by using a tool they implemented, *i.e.* StatSigMA-w. It is capable of evaluating the accuracy at each aligned site in multiple alignments and identifying suspiciously aligned regions, *i.e.* highly discordant and probably resulting from non-homologous regions.

StatSigMA-w extends the ideas of another tool, StatSigMA. Given a multiple alignment of k sequences and a phylogenetic tree for these sequences, StatSigMA computes a p-value for each of the k null hypothesis cases, *i.e.* one null hypothesis for each branch in the tree, stating that there is "unrelated behaviour" between the two subtrees separated by the removal of k (corresponding alignments are independent rather than homologous). If all of these hypothesis are rejected, one can say that all sequences are related. StatSigMA-w, on the other hand, performs this kind of analysis on every region of a multiple alignment. Their bench-test consists of 17 vertebrate alignments including the *human chromosome* 1, for which they identify 9.7% ($21Mbp$) of the alignment as being suspicious.

GRAPe method, assessment of homology at the nucleotide level Contrarily to (Prakash and Tompa, 2007) cited above, (Lunter et al., 2008) assumes that the regions of homology are correctly assigned and they focus on homology at the nucleotide level. Moreover, they deal with pairwise alignment only, as they explain that a thorough understanding of the pairwise case shall guide the design of methods for the multiple alignment problem.

For the sake of this study, pairwise DNA alignments of human and mouse were thoroughly examined. They reveal the existence of three types of alignment errors illustrated in Figure 2.5, *i.e.* *gap wander* caused by spurious high-sequence similarity in non-homologous regions, *gap attraction* happening with two indels having little separation and *gap annihilation* occurring when two indels have identical lengths and the two deletions are on different sequences.

(Lunter et al., 2008) prove that such alignment errors generate biases for both probabilistic and score based aligners, and depend only little on alignment parameters. Moreover, they observe that accuracy is the lowest in columns close to gaps and found that at least 15% of the aligned bases in the alignments were incorrect.

They also introduce a novel probabilistic alignment method, GRAPe, based on a standard three-state pair HMM (see page 25 in Section 1.6.2). GRAPe uses the posterior distribution of alignments to optimize the correct assignment of homology of

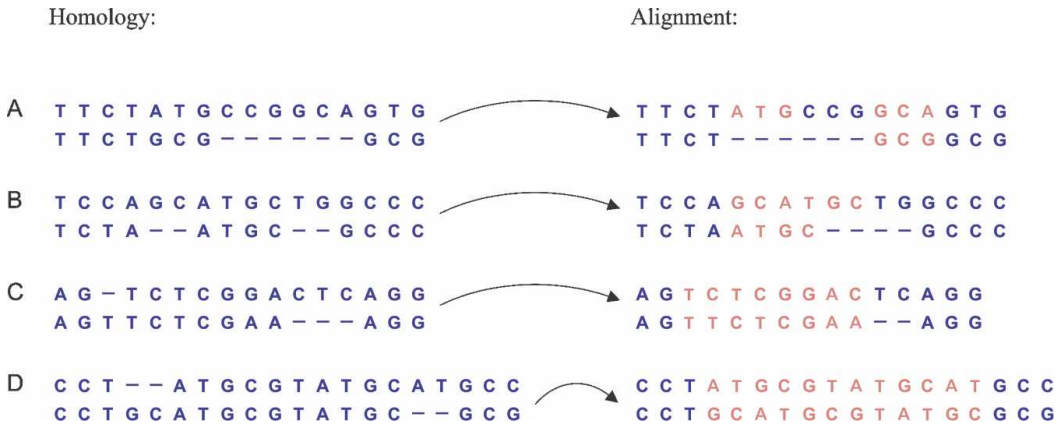


Figure 2.5: "Three types of alignment bias. Alignment algorithms are consistently biased toward likely distributions of indels across sequences, despite the occurrence of less-likely configurations at low frequencies. The figure shows four pairs of sequences with their homologies (left) and corresponding most-likely alignments (right), with wrongly aligned bases highlighted. We distinguish between three types of bias: gap wander (A), caused by spurious high-sequence similarity at nonhomologous sites; gap attraction (B,C), occurring when two indels have little separation, and gap annihilation (D), which occurs when two indels of equal size but opposite signature are found near to each other, favouring explanations without indel events." Figure extracted from (Lunter et al., 2008).

individual nucleotides, instead of finding a single most probable alignment. However, despite directed efforts for avoiding alignment errors, *e.g.* using probabilistic aligners, they concluded that alignment accuracy is "fundamentally limited". They estimate that for alignments of species at distances comparable to human and mouse, it seems likely that at least 10% of nucleotides in whole-genome alignments will remain wrongly aligned.

2.6.3 Comparative assessment of the quality of WGA tools

Comparative assessment of WGA in vertebrates In a very recent study, Martin Tompa and Xiaoyu Chen addressed different aspects on the comparison of alignment results (Chen and Tompa, 2010), by examining the results of four tools: TBA, MAVID, Multi-LAGAN and Pecan, on 28 vertebrate genomes. The test regions encompass 1% of the human genome and the homologous regions in 27 other vertebrate genomes. The alignment results, obtained by four expert teams from the ENCODE project after applying these four tools on a total of $554Mbp$ of genomic sequence, were analyzed by looking at the level of agreement between them, their coverage and their accuracy. A similar study can be found in (Margulies et al., 2007).

As to the problem of estimating accuracy, in an interview that can be found at [1], M. Tompa argues on the other two approaches commonly used: simulations and known features. He expresses very clearly his opinion on the use of simulated data, by calling this approach as "one way to get around the problem" because "if you simulate evolution then you do know what the ground truth is". In fact, as evolution is still incompletely understood, how come one is capable of modelling it? Moreover, M. Tompa explains that limiting the evaluation of alignments on known orthologous exons is not enough to draw a conclusion, as they do not extrapolate well to other genomic regions.

Thus, in order to assess accuracy, (Chen and Tompa, 2010) uses the method from (Prakash and Tompa, 2007), described above. The conclusion of their study was that there is a lack of general agreement between the four alignment methods, even on the well-known mouse genome. When plotting the coverage and the percentage of suspiciously aligned regions, Pecan emerges as the tool that obtains the best results on distant species. However, their study shows that building accurate multiple WGA remains an important challenge for non-coding regions and distantly related species.

Comparative assessment of WGA in bacteria In parallel with the work we conducted and that is described in chapters 3 and 6, an interesting study regarding the assessment of the quality of WGA in bacteria was published (Swidan and Shamir, 2009). Compared to (Chen and Tompa, 2010) who analyzed the complete alignment at the nucleotide level, the goal of their work was to evaluate the quality of the segmentation of the genomes with respect to the mosaic structure, see Section 2.1.1, and to quantitatively assess the quality of the alignment. For this, they introduce two types of measures based on:

1. *first*, how well the genome segmentation fits the gene annotation of the studied organisms, where annotations were obtained through KEGG (Kyoto Encyclopedia of Genes and Genomes) (Ogata et al., 1999). For this, they look for breakpoints of genes induced by the segmentation. Due to the high density of genes in bacterial genomes, breakpoints are taken into account only if located deeply inside genes. This should measure the level of imprecision of alignment tools as it reflects their tendency to miscalculate segment ends or to report erroneous correspondences between the segments.

2. *second*, the number of segments created by the alignment and the percentage of the two genomes that is conserved, *i.e.* a *whole genome percentage identity*.

Both types of measures are very simple, intuitive from a biological point of view, and easy to compute. Regarding the second type of measure, they argue that a segmentation with a greater number of parts allows for more freedom in the correspondence between them and therefore has a higher whole genome percentage identity. Moreover, they observe that the two types of measures are not independent and that they should be analyzed together. In a parallel study described in (Uricaru et al., 2009), we came

[1] http://www.genomeweb.com/informatics/qa-u-washingtons-martin-tompa-sizes-multiple-alignment-tools

to a similar result. The number of segments influences the number of disrupted genes, as more segments usually mean more breakpoints. Thus, based on the number of genes disrupted by the segmentation, a score was computed by normalizing according to the segmentation size.

These measures were used to estimate the quality of alignments for 41 intra-species bacterial pairs, given by two WGA tools especially designed for bacterial genomes: Mauve, described in Section 2.5.1, and MAGIC (Swidan et al., 2006), a tool published by one of the authors of this work. MAGIC is based on an intuitive clustering approach, which builds collinear blocks (similar to the LCBs of Mauve), while aiming towards identifying orthologous segments. As the evaluation relies on gene annotations, the alignment methods must be annotation independent. Thus, while MAGIC usually employs annotated genes as anchors, for this study anchors were provided by Mauve seeds. Regarding the disruption rate, the results of Mauve and MAGIC are close and significantly better than random. For the majority of pairs, MAGIC has smaller segmentation sizes and greater conserved percentages, while for the rest of them, the results were not conclusive. However, the fact that MAGIC obtains better results than Mauve in this evaluation setup is quite unsurprising, as it was designed to answer to this exact kind of question.

Conclusion These two recently published studies are important advances in the comparative, systematic assessment of the quality of WGA on both vertebrates and bacteria. However, additional work needs to be done in this direction, by comparing more tools on larger datasets based on highly accurate comparison criteria adapted to both coding and non-coding regions. Moreover, such comparative genomic studies revealed that closely-related bacteria often have highly divergent gene content and therefore WGA tools encounter difficulty in these cases. These directions of study are being pursued in Chapter 3.

3 Personal contribution. Whole genome alignment; applications to bacterial genomes

Unexpectedly, we found that the whole genome alignment of intra-species bacterial strains is incompletely solved nowadays. We begin this chapter with a detailed study on this matter. Then, we propose a new pairwise whole genome alignment strategy, less complex than classical WGA programs. This novel strategy is composed of two phases only: *fragment computation*, with fragments corresponding to local similarities (LS), combined with a *classical chaining* method.

Contents

3.1 Main contributions to WGA

In Chapter 2 we presented the latest achievements on the topic of whole genome alignment not relying on gene annotations. We have also discussed the drawbacks of existing methods and the potential directions of improvement in the field. Quoting W. Miller in 2001: *the genome alignment field needs improved datasets and protocols for evaluating the correctness and performance of genomic alignment software* (Miller, 2001). Although most genome alignment tools published since 2001 are variations on the anchor based strategy (see Section 2.2), few studies have analyzed the quality of alignments produced by such methods (Prakash and Tompa, 2007; Lunter et al., 2008; Chen and Tompa, 2010), and large benchmark datasets are still missing.

Moreover, even for the particular case of bacterial strains genomes, results are not convincing. In fact, things are far from being as simple as one may think. Recent efforts in systematic bacterial genomic comparisons revealed the need to improve methods and to build reliable resources in this field (Swidan and Shamir, 2009). Bacterial genomes from the same species are theoretically considered to be extremely close. The truth is that in practice, even intra-species genomes can be very different, due to their rapid evolution. Moreover, even animal and plant species are sometimes difficult to delimit, but microbes and especially prokaryotes seem to raise special problems. Their small size and general uncultivability (only a very small percentage can grow in the lab) complicate matters of description and classification of species (Konstantinidis and Tiedje, 2005).

Our work focuses on **pairwise whole genome alignment** applied to complete genomes of bacterial strains. We chose the pairwise aspect as it is easier than multiple alignment, and because it is the basic step in progressive multiple alignment. In fact, one needs reliable solutions for pairwise alignment before attacking the complex problem of multiple alignment. In the current chapter, we present our first main contributions to this field.

- *First*, we conducted an evaluation of anchor-based approaches for pairwise genome alignment, for the particular case of bacterial genomes. We describe the dataset in Section 3.2, and a protocol to evaluate the correctness and performance of such alignment tools from the computational point of view in Section 3.3. Our results were presented in parallel with those from (Swidan and Shamir, 2009) that pursued a similar goal: produce a protocol for evaluating the quality of the segmentation of the genomes and to quantitatively assess the quality of the alignment, in the case of bacterial genomes.

  Based on our initial evaluation of two state-of-art alignment tools, MGA (Hohl et al., 2002) and Mauve (Darling et al., 2004), we tend to contradict the common belief that the pairwise genome alignment problem in the case of bacteria is correctly solved nowadays. We identify two limitations of current methods: a **lack of sensitivity** generating the so-called **unsolved cases** and a **specificity problem** producing **incorrectly solved cases**. These results are discussed in Section 3.5.

Furthermore, we sought to determine which steps in the anchor based strategy were to be incriminated for incorrect or low quality alignments observed in certain instances. For this, we implement several pairwise alignment prototypes based on two steps: *fragments computation* and *chaining* (first and second phases of the anchor based strategy) and compare them to MGA and Mauve. These prototypes are described in Section 3.6. Experiments show that some of our prototypes manage to improve alignment results in divergent cases.

This preliminary work was presented in a national bioinformatics conference, *i.e. Jobim 2009*, (Uricaru et al., 2009).

- *Second*, thanks to the investigation described above, we concluded that: (i) the *last chance alignment* phase (fourth phase) introduces regions of low alignment, and (ii) sensitivity can be improved by exploiting local similarities as fragments. Therefore, we propose a less complex strategy than the classical one (composed of four phases). The innovative idea behind this novel strategy is the use of a fragment computation method that has never been used before in the first phase of anchor based tools: *long local similarities (LS)* obtained from spaced seeds.

  However, as explained later in this chapter, we realised that using local similarities in the first phase of the anchor based strategy requires an adapted chaining method allowing for overlaps. In Chapter 5 we define this novel problem of *chaining with proportional overlaps* and give an *exact algorithm* that we thoroughly describe and prove. Together with this novel chaining algorithm, we propose further on a new pairwise WGA tool, YOC, that was tested along with four other tools: MGA (Hohl et al., 2002), Mauve (Darling et al., 2004), ProgressiveMauve (Darling et al., 2010) and LAGAN (Brudno et al., 2003b), on a large dataset of intra-species couples of bacterial genomes, see Chapter 6. An article presenting these results is in preparation.

3.2 Datasets

Our work addresses the WGA in the particular case of the alignment of bacterial genomes. Knowing that most of the previous work related to the alignment of whole genomes was based on simulations, as discussed in the previous chapter, we decided to attack this problem from a different angle: *use real bacterial genomic sequences in order to perform thorough evaluations*. As data have evolved really fast in the last few years, our datasets also did. This is the reason why the results we present in this manuscript have been obtained on two datasets (one that we use for the experiments in this chapter, and a second one for those in Chapter 6), containing genomic sequences available in the GenomeReviews database (Sterk et al., 2006).

GenomeReviews database was chosen for a start because the database Mosaic (Chiapello et al., 2005; Chiapello et al., 2008) uses it as a resource and, as we shall see below, Mosaic is closely connected to our work. Second, and probably the most important aspect, is that GenomeReviews provides standardized and up-to-date genomes and

their annotations. These annotations were collected from several sources, including the EMBL Nucleotide Sequence Database, the UniProt Knowledgebase, the InterPro Protein Domain and Families database, the Gene Ontology Annotation Database, and others.

3.2.1 Mosaic database

The Mosaic db is a generalist comparative bacterial genome database, providing to the community easy access to comparisons of bacterial genomes at the intra-species level. It represents an excellent benchmark for bacterial genome comparisons, containing a complete set of alignments manually curated by experts, interpreted and classified, giving quality metrics that have been thoroughly tested and which can be easily reproduced. Thus, it is perfectly adapted to our needs, which consist in the evaluation of different WGA tools on pairs of bacterial genomes to point out the current limitations and to identify the tool that obtains the most accurate results.

MOSAIC updates regularly the bacterial genomes from the Genome Reviews (GR) database, meaning all genomes from the same bacterial species, according to the species names, which have more than two strains (without plasmids and extra chromosomal sequences). Thus, MOSAIC db evolved and it is now at its fifth version. For the comparisons, MOSAIC uses MGA [1] (Hohl et al., 2002) for cases without rearrangements, and Mauve [2] (Darling et al., 2004) for rearranged genomes. These two tools have been chosen for the Mosaic project as they are archetypes of WGA tools, especially suited to bacterial genomes, they cover the two types of comparisons: collinear and with rearrangements, they are fast and rather thoroughly described in the literature (see Section 2.5 for more details on these two tools).

In Mosaic, a comparison strategy was established and it was systematically applied on all intra-species pairs. First, as bacterial DNA is often circular, false rearrangements may be induced by different cutting points. Therefore, Mosaic applies a pre-processing step that consists in shifting sequences in order to eliminate such problems. Next, the shortest of the genomes in the pair of genomes is chosen as reference. Mauve is applied, with parameters that have been adjusted on *E.coli* strains and tested on *Shigella flexneri* strains, to detect macro rearrangements. It is the number and the size of Mauve LCBs that provide information on the existence of rearrangements (see Mauve details in Section 2.5). If no macro rearrangements were detected, the pair of strains is considered as being *collinear*. In the case of collinear genomes, MOSAIC uses MGA, which has been observed as being more accurate on genomes without rearrangements.

The complete Mauve parameters settings are as follows: *seed-size* = 19, *island-size* = 20, *backbone-size* = 20, *max-backbone_gap* = 20, *gapped-aligner* = *clustal*, *max-gapped-aligner-length* = 10000, *min-recursive-gap-length* = 5000, and *weight* = 5000. MGA parameters were set up as follows: $l = 50 - 20$ and $gl = 3000$. The ini-

[1] MGA version 2003-03-18
[2] Mauve version 1.2.3

tial seed size limit was fixed to 19 as to correspond to the match size of 20 used by MGA during the recursion. The *max-gapped-aligner-length* parameter represents the maximum number of base pairs to attempt aligning with the gapped aligner, *i.e.* ClustalW, producing the so-called *aligned gaps*, while the *weight* refers to the minimum LCB weight in base pairs.

MGA and Mauve raw alignment results could have been used to directly define backbone and variable segments (see page 37, Section 2.1.3). However, it has been observed that such results are of unsatisfying quality (Chiapello et al., 2005). Moreover, a recent study (Devillers et al., 2010) meant to measure the robustness of bacterial genome segmentations, applied on the alignments included in the Mosaic db, revealed the presence of biologically irrelevant backbone and variable segments and insisted on the necessity of clearly identifying the limitations of WGA tools before using them on a particular set of genomes. Thus, Mosaic employs a strategy for post-processing alignments in order to filter low quality parts and precisely identify more reliable backbone and variable segments. An accurate identification of such segments allows comparative and evolutionary analyzes of both coding and non-coding regions of bacterial genomes. The post-processing strategy consists of a filtering step as follows. Based on the initial alignment, matches are always kept in the backbone, while aligned gaps, added in the *fourth phase* of the anchor based strategy (see page 41, Section 2.2), inferior to 76% identity (id% calibrated on *E. coli*) are considered to be variable segments, together with insertions and unaligned regions.

MOSAIC computes several metrics, including the *backbone coverage*, *i.e.* the total length of the regions considered to be part of the backbone, the *identity percentage of the backbone*, *the number* and *the size* of segments that are part of the backbone. Similar metrics are described in the next section.

Comparisons giving as result too low coverage backbones are not included in the MOSAIC database, *i.e.* backbone coverages smaller than 50% of the average length of genomes. Five bacterial species, i.e. *Buchnera aphidicola*, *Prochlorococcus marinus*, *Pseudomonas fluorescens*, *Rhodopseudomonas palustris* and *Synechococcus sp* were thus excluded due to **poor quality results**. In fact, they either represent unusually divergent bacterial genomes, or they are part of a genus composed of several species.

Dataset 1 The experiments described in this chapter, corresponding to the first part of my thesis, were carried out on a preliminary dataset, to which we shall refer to as *dataset* 1. The *dataset* 1 consists of all 236 pairs of bacterial strains of the same species (140 different genomes), coming from 42 different species having at least two complete genomes available in Genome Reviews database at mid-2008, as in the release 4.0 of the Mosaic database (Chiapello et al., 2008).

- 199 pairs (84%) were inserted in the MOSAIC database according to the criterion of 50% of minimum coverage of the backbone regions compared to the average length of genomes;
- 95 pairs (40%) were considered as being collinear according to the criteria described in (Chiapello et al., 2008), *i.e.* no inversion or translocation were detected

with Mauve aligner, or none of the inverted or translocated segments detected by Mauve exceeded a threshold of 20 kb in length.

3.3 Global qualitative and quantitative criteria for assessing the quality of alignments

Newly implemented methods need to be compared to existing methods and alignment results need to have their quality estimated. However, as discussed in the previous chapter, a comparison protocol for global alignment has not yet been established.

The underlying biological question is which regions are common to both genomes and which are specific. An alignment defines a partition in common and specific regions. Precisely comparing two solutions would require to examine the boundaries of each region and its local alignment in each solution. Such a detailed comparison procedure is too long, at least for a systematic application, and inappropriate for genome alignments that often disagree on many regions.

We need global quantitative criteria to assess the quality of alignments and compare several programs on a large number of test cases. Below, we define two criteria: the *coverage*, and the *identity percentage*, two global measures of alignments. Many tools compute values of coverage and identity percentage, but not necessarily with the same definition. First, one thing has to be made clear. In this chapter we refer to the *backbone* as being *the complete set of regions common to the compared sequences, in the way that they are given by the WGA alignment*. It is on the backbone that we compute the *coverage* and the *identity percentage*. On the other hand, the results presented in Chapter 6 are based on filtered backbones, *i.e.* obtained after applying the filtering procedure described in Section 3.4.

Coverage We define the *coverage* as the total length of the set of regions that are common to both sequences, *i.e.* the size of the backbone. We name *coverage%* (*cov%*) the ratio between the coverage and the genome length. Both the coverage and the coverage% are being computed for each of the compared sequences.

The identity percentage is a well accepted measure of the alignment quality. It is usually computed as the number of identical base pairs over the total alignment length (*e.g.*, as in BLAST). However, this makes the id% incomparable between alignments that differ in length, which is the case with LS. Next, we give the definition of the id% as employed in this chapter.

Identity percentage We define the *identity percentage* (id%) to be the ratio of identical base pairs in the segments of the backbone over the genome length. Hence, at its maximum, the id% equals the coverage, *i.e.*, when all aligned segments are identical. Thus, the id% measures the quality of the genomes' parts that can be aligned to each other, and is comparable across genome pairs. Same as for the coverage, we compute one id% value for each of the compared sequences .

The id% also refers to: *first*, the **identity percentage of a segment**, *i.e.* the ratio of identical base pairs over the length of the alignment corresponding to the segment, and *second*, the **identity percentage of the coverage**, *i.e.* the ratio of identical base pairs in the segments of the backbone over the total length of the backbone. This latter measure can be computed by dividing the id% by the coverage%.

Correlations between measures Defining the id% in relation to the length of the aligned regions makes the id% be in complete correlation with the coverage, and not explainable in the absence of the latter. However, it sharply estimates the quality and the relevance of the aligned regions. On the contrary, defining the id% in relation with the length of the genomic sequences, as in our case, makes it an independent measure. This is an advantage when we need it for comparing different pairwise alignment results. As our goal is to establish metrics that would help us compare different alignment methods, we prefer to define the identity percentage in the second manner.

Basically, high coverage and high id% is the perfect alignment case: a complete, good quality alignment. High coverage and low id% means that most of the coverage is given by low quality aligned regions and thus, we are probably dealing with very divergent genomes. In this case, we suspect part of the anchors to be false positives. If the coverage is low and we have an almost equal id%, it means that we probably got good anchors but we deal with a difficult case (every tool has its own specific difficult cases). This is probably the case of rearranged and/or divergent genomes if we work with tools that don't deal with rearrangements, and divergent genomes for tools that treat rearrangements. Low coverage and low id% is clearly an unsolved case. This means that whether we deal with genomes that have nothing in common, or we find ourselves in a case that is not correctly treated by the tool that we are using.

Moreover, we can discuss the **complexity of the segmentation**, *i.e.* the number of segments splitting up the backbone. Both the id% and the coverage are closely connected to the complexity of the segmentation but the number of segments makes sense only if the coverage and id% are not extremely small. A partition in a great number of segments means difficult readability of the alignment and an unclear knowledge of the genome evolution. Therefore, we prefer a partition of the alignment in a small number of long segments. Exact matches, like in MGA, determine a big number of segments. Approximate matches and especially local alignments, simplify the alignment segment map.

3.4 GRAPe filtering procedure

As to estimate the quality of aligned segments, we applied GRAPe (Lunter et al., 2008), a probabilistic genome aligner capable of quantifying the uncertainty of every position in a given alignment, see Section 2.6.2.

GRAPe is applied on a piece of (or on a complete) pairwise alignment obtained by an alignment tool. Because of its inability to deal with long sequences, we proceeded to the fragmentation of alignments in 500 length pieces and used GRAPe to realign these pairs of short sequences. GRAPe parameters were fixed so as to cope with divergent sequences, *i.e.* $d = 1000$ and $a = 1000$.

Lines in the alignments output by GRAPe were accompanied by an additional line annotating the posterior probability of every column. The posterior probabilities range from 0 to 1, represented by letters a-zA-Z, where a=0 and Z=1. For ungapped columns, it refers to the posterior probability that those nucleotides align. For gapped columns, the number represents the posterior probability of the nucleotide not being aligned to any nucleotide. See Figure 3.1 for an example of a GRAPe alignment result.

```
361 chrX 8437003 8437070 chr8 70513702 70513775 + 1992
AGAC--------TGCCCGTGCATATATACCAGTTACTATATGGACAGTTAAAAAAAATAGGGAGAGAGAAATCAAT
AGAAGAGCAATATGTTTGTACACATATTCCTGTTACTTTATGAACAATA--AAGAAATGGGAACAGGGAAATGAAG
hhihKMNNOONKtPTUVXYZZZZZZZZZZZZZZZZZZZZZZZZZZZZZXREBBEHKORVYYYYYYYYYYYYXWVQHEx
```

Figure 3.1: GRAPe alignment output: the two alignment lines are followed by a line annotating the posterior probability of every column, with probabilities represented by letters a-zA-Z, where a=0 and Z=1.

Thus, the novel alignments produced by GRAPe and the probabilities associated to them were interpreted as it follows:

- positions in the alignment that were aligned with a probability inferior to 0.05 were considered as being "unalignable". If WGA tools aligned these position, then they are considered to be *false positives*, and if not, they represent *true negatives*.

- positions in the alignment inside insertions/gaps, having a probability superior to 0.95 of being a true insertion position, were also considered as being "unalignable". Same as in the previous case, if WGA tools aligned these position, then they are considered to be *false positives*, and if not, they represent *true negatives*.

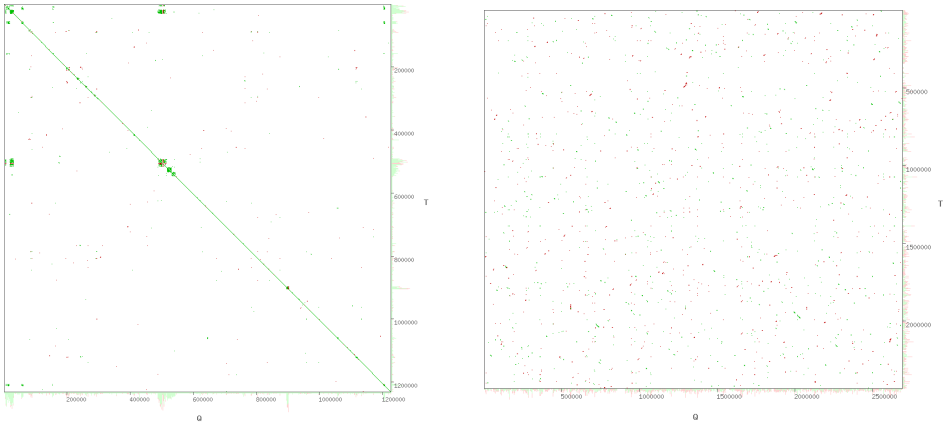
- 500 length pieces of alignments having more than half of their positions "unalignable" were removed, *i.e.* filtered, from the alignment.

3.5 Initial assessment of alignments provided by two state-of-the-art WGA tools

Before working on an improved solution for the pairwise whole genome alignment problem, we did an initial analysis of the alignments provided by MGA and Mauve on a set of intra-species pairs of bacteria (*dataset* 1), in order to identify their limitations. MGA and Mauve are described in Section 2.5.1. Here, we summarize their main characteristics: in the *first phase*, MGA uses *MEMs*, while Mauve uses *approximate matches*, in the *second phase*, MGA computes a *collinear* chain and Mauve, an *unconstrained* set of anchors; the *third* and the *fourth* phases are very resembling. Thus, both tools are using short fragments, they do not allow overlaps and use recursion and *last chance alignment*.

Parameters for the two tools, described in Section 3.2, have been retrieved from the release 4.0 of the Mosaic db. Note that we are working with the raw alignments, meaning alignments before the Mosaic specific post-processing step, and that low quality alignments, not included in Mosaic db, have also been taken into account in our study.

As to assess the quality of alignments and to compare the alignments given by the two tools for the same sequences, we used the two measures: cov% and id% (detailed in Section 3.3). To be certain that we are measuring the same things, we needed to adapt the computation of these measures to the specific output of each of these two tools.



(a) (b)

Figure 3.2: Dot plots representing pairwise local alignments of complete genomes, obtained with YASS software, described in Section 3.6. (a) Alignment of two very close strains of *Chlamydia pneum.* species. (b) Alignment of two strains of *Synechococcus sp.* that seem to be completely unrelated, thus unalignable by any alignment tool.

The segments taken into account in the coverage, for both tools, correspond to the

anchors selected in the second and third phases of the anchor based strategy, together with the *aligned gaps* added in the fourth phases, see page 41, Section 2.2. In our definition of coverage, unlike the coverage computed by MGA and Mauve, insertions are not taken into account.

As for the id%, we can exactly compute it for MGA alignments, based on the segments taken in the coverage. For Mauve however, we know that the version used for the alignments in Mosaic employs some sort of spaced seeds for detecting anchors, thus anchors are not necessarily exact matches like in MGA's case. As the alignments of the anchors are not included in Mauve output, it is not possible to obtain their id% without realigning them, which we did not do. Thus, for the id% of Mauve alignments, we took anchors as if they were exact matches. This leads to an overestimation of the true id%, however small, as in practice anchors are short and have few mismatches.

3.5.1 Discussion on results

When examining the alignments, the first striking result lies in the difference of coverage between different species obtained by both tools. For some species, all pairwise alignments cover more than 90% of the genome (*e.g.*, *Streptococcus thermophilus*), while for others the cov% is below 10% (*e.g.*, *Synechococcus sp*). We also observe, but more rarely, species for which the cov% of both methods varies greatly among pairs of strains (for *Prochlorococcus marinus*, the cov% of MGA ranges in $[0, 78]$% and that of Mauve in $[6, 96]$%). It is clear that these programs succeed in aligning some genome pairs and fail on others.

The fact that tools fail to align some of the pairs could be due either to a high level of divergence that makes the sequences unalignable (a biological explanation), see Figure 3.2, or to a methodological failure in detecting the regions of similarity or in chaining. Moreover, it is true that calibrating parameters on *E. coli* (not a very divergent bacteria), could explain the unsatisfactory results obtained for species as *Buchnera aphidicola*, *Prochlorococcus marinus*, *Pseudomonas fluorescens*, *Rhodopseudomonas palustris* and *Synechococcus sp*, which were not included in the Mosaic db. In fact, it is known that even though the rate of recombination in bacteria is not as high as that of animals, it can still be very high, several bacterial species present superior divergence rates than others (*e.g.*, *Buchnera aphidicola*) and others are in fact genus composed of several species (*e.g.*, *Synechococcus sp*). Therefore, the choice of parameters should be adapted to the individual case of each species.

However, if unalignable sequences due to biological causes are likely to stay that way in the future, one should try to improve solutions in the remaining cases. If we leave the biological causes of unalignment aside and we focus only on methodological failures, there are two possible explanations for the low quality results: first, they may be **incorrectly solved cases** and second, they may be presently **unsolved cases**, *i.e.* for which, tools are not yet able to give a solution. However, the frontier between these two possible explanations is not always very clearly drawn.

Figure 3.3: A 1298 bp Mauve alignment piece in an aligned gap of a *P. marinus* couple of strains.

3.5.2 Incorrectly solved cases – specificity problem

Incorrectly solved cases are mostly due to methodological limitations that do not need further explanation, *e.g.* tools that do not deal with rearrangements produce low quality results on non-collinear pairs. However, a large number of incorrectly solved cases are generated by the use of improper procedures at some point of the alignment process.

An example of what we consider to be such an incorrectly solved case is the *P. marinus* pair, $CP000111$ vs $CP000095$, a species known for its divergence and its abundance of rearrangements. For this couple, MGA obtains 4% of coverage%; this low result can be easily explained by the fact that MGA was designed for close, collinear genomes, thus it cannot possibly deal with this pair of strains that has undergone multiple rearrangements. In fact, from the point of view of MGA, this case is practically an unsolved case, like those we discuss in the following paragraph.

Mauve, on the other hand, was especially thought for dealing with this kind of complex cases, therefore it reaches 84% of coverage%, however with only 45% of id%. This can be interpreted as Mauve covering the genomes with segments that have in average 54% of identities. Indeed, for the same pair, when plotting the cumulative cov% of segments with the % of identities below a given threshold, see Figure 3.4, we find that Mauve covers 22, respectively 30% of the genomes, with segments that have ≤ 50, resp. $\leq 55\%$ of identities.

When looking into Mauve's alignment, we observe that it is composed of $75Kbp$ of anchors and $1400Kbp$ corresponding to 2926 "last chance alignment" segments, *i.e.* aligned gaps (see page 41). As aligned gaps make most of Mauve alignment,

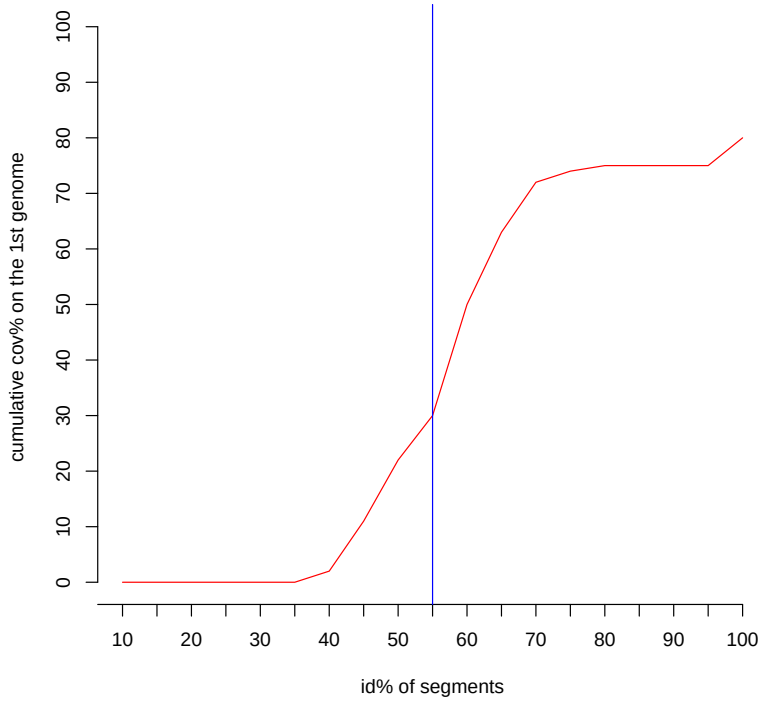


Figure 3.4: Cumulative cov% on levels of id% for Mauve backbone segments on *P. marinus* pair: $CP000111$ vs $CP000095$. Mauve covers 22, respectively 30% of the genomes, with segments that have ≤ 50, respectively $\leq 55\%$ of identities.

we examine them closer. For example, if we take a piece of such aligned gap, as in Figure 3.3, we notice its extreme low quality.

Further on, we applied GRAPe (Lunter et al., 2008), a probabilistic genome aligner capable of quantifying the uncertainty of each position in the alignments, see Section 3.4 for a complete explanation on the protocol established for using GRAPe. In average, on the 2926 segments, GRAPe points out 13% of positions as unalignable with more than 0.95 probability. Moreover, with the same probability threshold, 12% of segments ($= 351$) have $> 50\%$ of unalignable positions; in average on the two genomes, these segments represent $516Kbp$ of alignment, meaning a quarter of the total Mauve alignment. Based on these observations, we suppose that part of the aligned gaps should not be taken into the final alignment.

In conclusion we can make the following remarks on this kind of incorrectly solved cases that perfectly concord with (Chiapello et al., 2008), highlighting the necessity of improving on the sensitivity and the reliability of methods like MGA and Mauve:

- are likely to appear in *divergent sequences*;

- they contain *false positives*, *i.e.* alignments of unrelated regions;
- are mostly produced by the *"last chance alignment"* phase.

A straightforward solution to avoid taking unrelated regions in the alignment and to avoid in this manner incorrect solutions, would be to post-process alignments by filtering the alignment pieces, *e.g.* in GRAPe's manner (Darling et al., 2010). Unfortunately this kind of solution is highly computationally expensive and is based on the fact that the alignment is globally correct, meaning the order of pieces. A similar solution, *i.e.* filtering based on a id% threshold, however simpler and less computationally expensive, is more arbitrary and difficult to adjust on each particular case (Chiapello et al., 2008). Thus these solutions are of limited help and, moreover, we consider that the pursued goal should be to directly produce correct alignments and not to filter them.

3.5.3 Unsolved cases – sensitivity problem

What we name unsolved cases, often corresponds to couples having an extremely high level of divergence for the intra-species level. Surprisingly often, these couples are collinear couples.

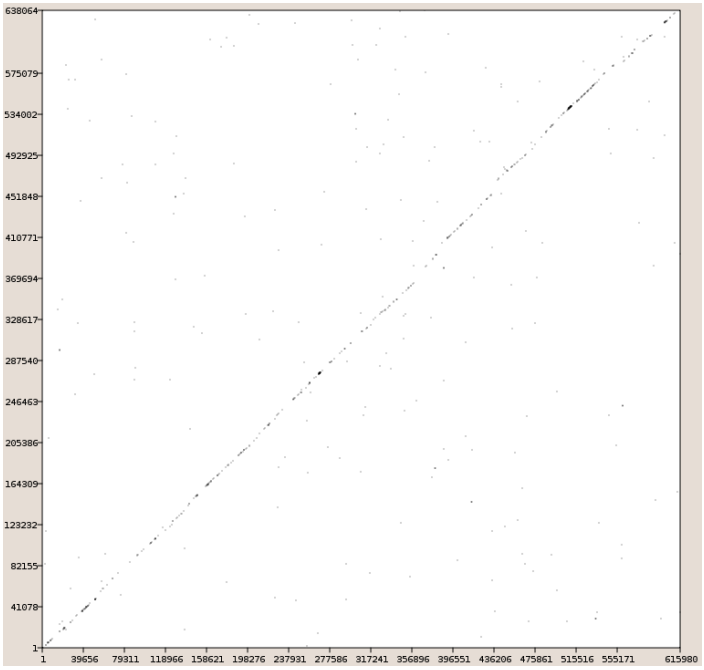


Figure 3.5: 20*bp* MEMs on *Buchnera aphidicola* pair, *AE*016826 vs *BA*000003, obtained in the first phase of MGA, before chaining.

In Figure 3.5 we show the plot of the alignment obtained by MGA (with MEMs) for the *Buchn. aphidicola* pair, *AE*016826 vs *BA*000003. As it is clear from the image, the *Buchn. aphidicola* pair is perfectly collinear but it suffers from a high level of

divergence, *i.e.* observe the numerous discontinuities in the main diagonal of the plot. Even though it may seem easy to build an alignment following the main diagonal in the plot, in practice tools give very poor results for this case. If we examine the MGA alignment for example, we learn that it covers 35% from the genomes ($\approx 220Kbp$), partitioned in 800 pieces (an average of $0.2Kbp$ by piece). This is a clear example of a "false negative", *i.e.* an unsolved case due to methodological limitations that could be highly improved once the sensitivity of the methods is improved.

3.6 Initial study of the impact of using local similarities as fragments

In this section, we propose a solution for both, increased specificity and sensitivity: *the use of local similarities (LS)* instead of exact/inexact matches. The *fragments computation* phase is mainly responsible for the sensitivity of anchor based methods. Indeed, the chaining phase only discards potential anchors. So in order to improve the quality of such methods, the first idea is to work on the sensitivity of this initial phase. We thus suggested to replace methods computing short exact (or approximate) matches with seed-and-extend methods that generate local similarities, see Figure 3.7. *First*, local similarities are capable of detecting larger similarity regions that are more likely to make biological sense. *Second*, seed-and-extend methods are more adapted to divergent sequences, finding significant similarity between sequences where short, exact matches may be too rare.

In order to show that using large similarity regions truly improves the results of the anchor based strategy, we had to assess the impact of different types of fragments on the sensitivity of this strategy. For this, we tested several fragment computation methods producing either short exact matches or local similarities, together with a classical fragment chaining phase.

To generate exact matches we chose the classical MEM computing tool: VMatch[3] (Kurtz, 2003), like in the first phase of MGA (see Section 2.3). For local similarities, we chose two different seed-and-extend methods: the well-known Gapped-Blast [4] (Altschul et al., 1997) for nucleotide sequences based on contiguous seeds (see Section 1.6.1), and YASS[5] (Noé and Kucherov, 2005), a fast similarity search program based on spaced seeds, described below.

Let us look first at the *Buchn. aphidicola* couple in Figure 3.5. For this same couple, in Figure 3.6, we plot the LS obtained with YASS. The difference between the two images is flagrant, as we can easily see that LS manage to perfectly cover the main diagonal, with much less pieces than when using MEMs. This is an important clue pointing to the hypothesis that LS help improving sensitivity of WGA.

[3] VMatch version 2004-03-10
[4] Blast version 2.2.18
[5] YASS version 1.14

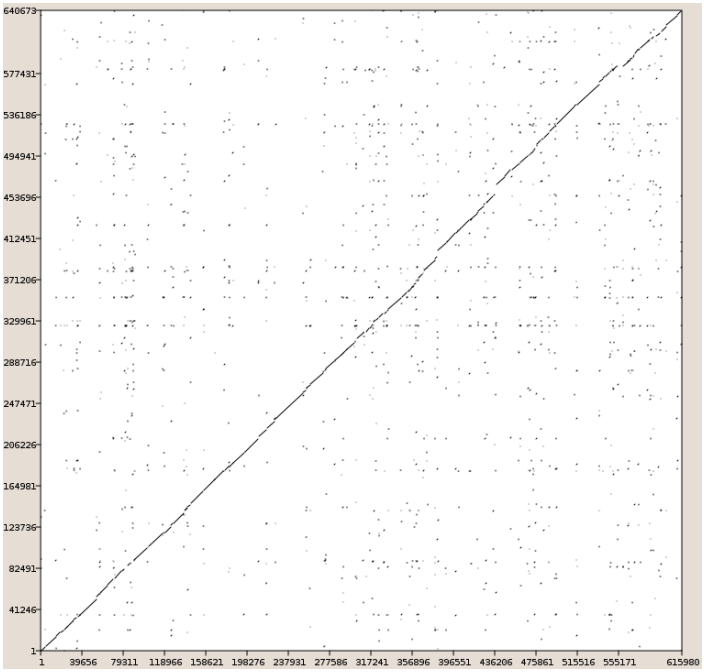


Figure 3.6: LS obtained with YASS on *Buchnera aphidicola* pair, *AE*016826 vs *BA*000003.

3.6.1 YASS, a similarity search program based on spaced seeds

YASS (Noé and Kucherov, 2005) is a DNA pairwise local alignment tool based on an efficient and sensitive filtering algorithm that uses a flexible hit criterion in order to identify groups of seeds. Compared to the classical heuristic alignment tools (*e.g.*, Blast, Fasta), which require an exactly matching k-mer, YASS uses the *spaced seeds technique* (Ma et al., 2002) that allows an increase in sensitivity without loss in selectivity. Moreover, YASS uses a *transition-constrained seeds model*, which capitalizes on the statistical properties of real genomic sequences. Based on a statistically founded *hit criterion*, it searches for closely located seeds, likely to belong to the same alignment.

Spaced seed patterns and seeds *Seeds* are formed by two words, one from each sequence, which match a seed pattern. A *spaced seed pattern* is built over a three-letter alphabet #, @ and –, where # stands for a nucleotide match, – for a "do not care symbol" and @ for a match or a transition.

The selectivity of a seed depends on its weight, defined as the number of # and half the number of @ characters. Less # and @ characters the seed contains, the higher its sensitivity. In order to assess the sensitivity of a given seed, one may use the approach described in (Kucherov et al., 2006), based on a Bernoulli model, simulating alignments of non-coding DNA, and an HMM, for alignments of coding DNA.

For a given seed pattern, the program identifies seeds, which match at positions

specified by the pattern. A seed occurring in the two sequences, means a hit of a potential alignment. The search for seeds is being done on both strands. Moreover, one may use groups of seed patterns, instead of a single pattern to increase the sensitivity. Next, YASS extends the seeds in order to form groups of seeds (inside which seeds may overlap) that respect several criteria related to the number of matches in the group, with the inter-seed distance and the seed diagonal distance.

Comparative experiments show that with equal selectivity level and lower running time, YASS reaches better sensitivity than traditional approaches like Gapped-Blast. YASS detects similarities that cover about twice the overall length of those found by Gapped-Blast, while it keeps only local alignments with E-values below 10^{-6} (Noé and Kucherov, 2005).

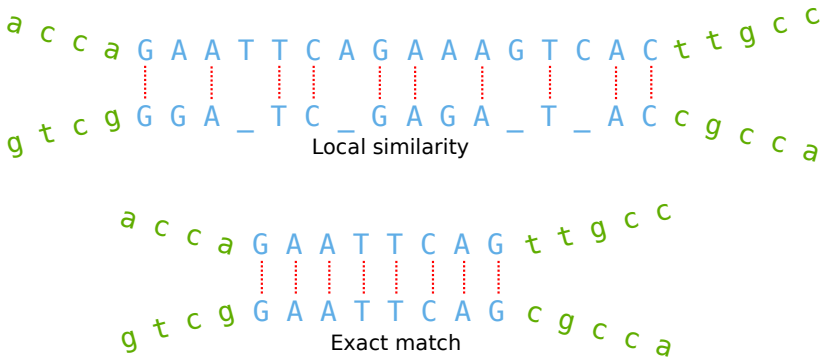


Figure 3.7: An example of local similarity versus an exact match.

3.6.2 Experimentation protocol

These three fragments computation methods: VMatch, Blast v.2 and YASS are combined with Chainer, a tool described in Section 4.9, implementing the classical consistent chaining defined in Section 4.3. **We thus obtain three prototypes to test: VC for VMatch and Chainer, BC for Blast and Chainer and YC for YASS and Chainer, see Figure 3.8.**

For each of the four tools we tested multiple parameter settings. We were seeking for parameter values that would allow programs to stay fast but in the same time to be the most sensitive possible. In fact, as we only used the first two phases of the anchor strategy, the idea was to let the chaining phase do the filtering, instead of doing it directly from the first phase. This should allow the detection of distant similarities.

For all three computing fragment tools, we looked for matches on both strands in order to detect both forward and reverse fragments (local inversions). Matches on the reverse strand were reversed and projected on the direct strand in order to be taken into account in the chaining process, *i.e.* remember that the classical chaining is collinear, thus fragments need to be in the the same direction on both strains.

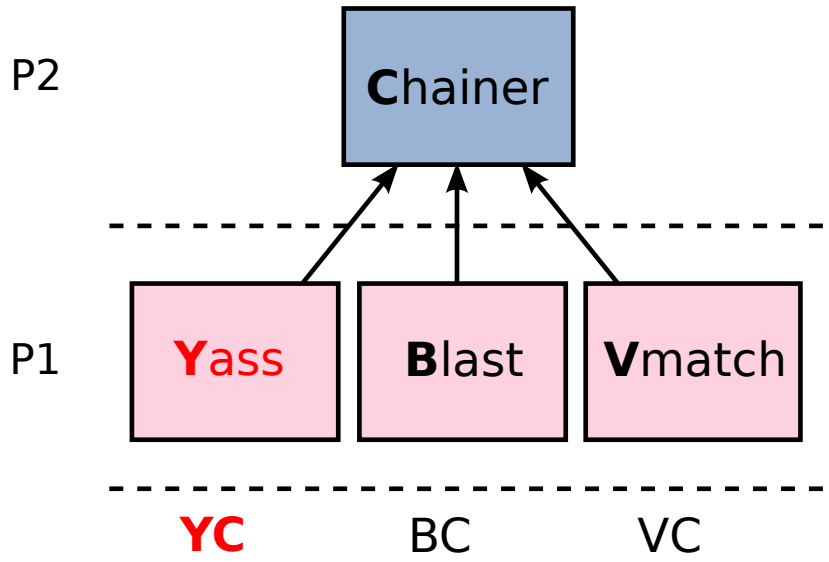


Figure 3.8: Three prototypes composed of two phases, *i.e. fragments computation + chaining*, namely: VC for VMatch + Chainer, BC for Blast + Chainer and YC for YASS + Chainer.

VMatch We fixed the minimum match size to 20*bp*. In fact, choosing a threshold smaller than 20*bp* slows the computation process and does not improve the global results (as tests for 10*bp* match size showed). On the other hand, by using a higher threshold we would miss similarity regions in more divergent sequences: tests with 30*bp* showed that results were sensibly in favour of the 20*bp* value.

YASS We fixed the E-value threshold to 10, and kept the default values for the other parameters, *e.g.* DNA matrix: $-C$ 5, −4, −2, −4, −4, 5, −4, −2, −2, −4, 5, −4, −4, −2, −4, 5. The set of seeds we used is $-p$ "#@_##_## _#__@_###, #_##@____# #____#____#@#_#". These seed patterns are a default choice for YASS; they were designed by *Iedera* program (Kucherov et al., 2007) and are meant to be a fair compromise for the sensitivity in both coding and non-coding regions. We tested several E-value thresholds: 0.1, 1, 10, the first two ones limiting the number of local alignments. The differences in cov% revealed to be, with the exception of very few cases, insignificant. By choosing the E-value threshold to be very permissive, we hoped to detect fairly divergent sequences. This strategy pays in the case of *B. aphidicola*, a divergent species not included in Mosaic db. In fact, for this case, the coverage obtained by YASS+Chainer with E-value 10 outperforms by 7% the one given by YASS+Chainer with both E-values 0.1 and 1. However, for less divergent cases, the results are extremely close.

Blast We used the Blast version for nucleotide sequences with default parameter settings, except for the E-value that we fixed to 10 as in the case of YASS.

Chainer We employ the *global strategy of chaining* (instead of the local one), as it is more appropriate when building a WGA. Fragment weights were given by fragment lengths in the case of exact matches produced by VMatch. For YASS and Blast, as they output non-exact matches, we had to choose between several possibilities: their length, the number of their identical base pairs and their score. As the results were slightly better when using the number of identical base pairs, we finally kept this metric. Moreover, in this way we merge the advantage of the other two ones: length and quality.

These prototypes, VC, BC and YC, are in fact three WGA methods, based on a simple two-phase strategy, *i.e.* without the third and fourth phases classically implemented in the anchor based strategy. It is straightforward that if we want a precise examination of the effect of solely the fragment computation phase, we do not need the last two phases.

In Section 3.6.3, we test the three prototypes on *dataset* 1. For each strain pair in the dataset, we computed the differences of cov% and id% between the method that detects exact matches (VC) versus the two methods based on local similarities (BC and YC). In Figure 3.9 we summarize the distribution of these differences using the classical box plot representation.

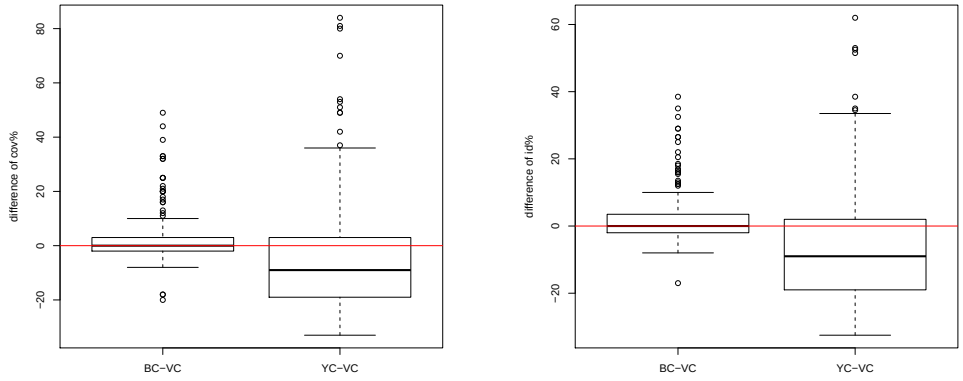
Box plots In statistics, a box plot, also known as a box-and-whisker plot, is a convenient way of describing groups of numerical data through their five-number summaries: the smallest observation (sample minimum), lower quartile (Q1), median (Q2), upper quartile (Q3), and largest observation (sample maximum). A box plot may also indicate which observations, if any, might be considered outliers. Outliers usually indicated by open dots are, as in our case, those observations that lie more than 1.5 times the interquartile distance below or above the first and third quartiles respectively, see for example Figure 3.9.

3.6.3 Fragments computation sensitivity analysis

We would normally expect the spaced seed method, YC, and the Blast approach, BC, to obtain better results than the MEM based approach, VC. In Figure 3.9, we can observe that this is not the case. For instance, the median of (YC-VC) for both the cov% and the id% lies around -10%.

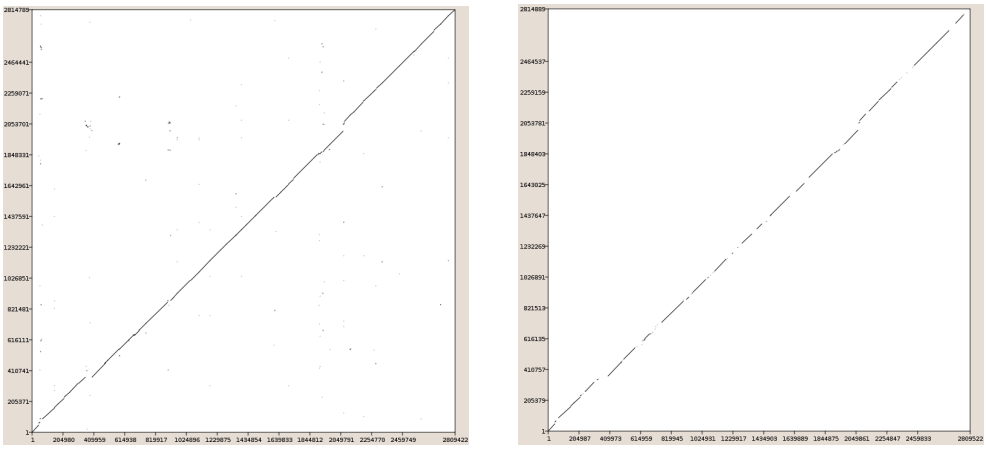
An analysis of YC alignments showed that, surprisingly, the worst results were obtained for pairs of close collinear genomes covered with long local similarities: **YC was unable to include some of them in the chain because of overlaps**. In such cases, VC detects multiple short fragments and chains them well. This suggested that the classical chaining algorithm (Chainer) is unadapted to local similarities. Indeed, this definition of a chain prohibits overlaps between adjacent anchors, see Figure 3.11.

For example, when comparing strains $CP000046$ and $BA000018$ of *Staph. aureus*, YC obtains a cov% of 65%, while VC reaches 88%. Figure 3.10 perfectly depicts



(a) (b)

Figure 3.9: Box plots representing the distribution of cov% (a) and id% (b) differences, corresponding to the alignments of pairs of strains in the dataset, for VC vs BC and VC vs YC. BC obtains better or similar results to VC in a big number of cases. YC improves on VC in some cases, but in most of the cases it obtains surprisingly bad results, *i.e.* median close to −10 for both cov% and id%.



(a) (b)

Figure 3.10: Dot plots for $CP000046$ vs $BA000018$ of *Staph. aureus* showing LS obtained with YASS in (a) (before chaining) and the same LS after the chaining phase in (b), showing the loss of quality between the two phases.

the reason of such poor result obtained with LS, as it shows LS before and after the chaining step, in order to observe the loss of quality from one step to another. In fact, YC's chain is interrupted by 17 holes of more than $10Kbp$ each. For 14 of these holes, at least one large local similarity (average size $37Kbp$) was not included in the chain, because of an overlap with an adjacent segment on one or both genomes. All overlaps measure between $1bp$ and $1.8Kbp$ in length, with an average at $218bp$.

We observed that short overlaps are usually due to randomness. The probability of having the same characters at the extremities of the fragments is high, due to the small size of the DNA alphabet, which consists only of 4 characters. The lack of precision at the extremities of fragments, caused by the extension strategies in seed-and-extend approaches, also leads to short overlaps. Large overlaps on the other hand, are due to variable tandem repeat structures that differ in number of copies between the strains or that can be found in only one of the strains. If one would want to correctly align such structures without breaking the region in two overlapping fragments, one would require a more general alignment model and specific algorithms (Bérard and Rivals, 2003).

Figure 3.11: An example of fragment not included in the chain due to overlaps on two adjacent fragments.

3.7 Conclusion

As discussed in Section 3.1, bacterial genomes from the same species, theoretically considered as being extremely close, can be very divergent in practice. In Section 3.5, we observe that *tools fail to align some of the pairs*, mainly due to specificity and sensitivity problems.

In Section 3.6, we suggest *replacing, in the anchor based strategy, methods computing short exact (or approximate) matches with seed-and-extend methods that generate local similarities*. Based on initial experiments, we observe that local similarities have a specific problem: *frequent overlaps of various sizes*. Moreover, we show that overlaps' lengths cannot be easily bounded by a constant. Thus in order to take overlaps between fragments in a chain into account, one should develop solutions that allow overlaps with maximal accepted lengths related to the lengths of the fragments. In Chapter 6 we introduce a novel definition of collinear chaining, allowing for overlaps proportional with the lengths of the adjacent fragment. For this novel problem, we propose an exact algorithm called OverlapChainer that we thoroughly describe and prove. In Chapter 6, we employ this novel chaining method for the WGA of bacterial genomes.

4 State-of-the-art. Preliminaries, formulations and solutions for chaining

To cope with the large volume of data, software tools designed for whole genome alignment use an anchor-based approach composed of three phases. The current chapter is dedicated to the second phase of this approach: *fragment chaining*. It consists of a thorough analysis of the different versions of the chaining problem, from the classical *collinear chaining* to the complex *chaining with rearrangements*, passing through the more novel approach of *glocal chaining*.

Contents

4.1 Introduction to the problem of chaining fragments

Several problems encountered in computational biology are special cases of the fragment chaining problem. An example of such problem is mapping a large number of cDNAs (mRNAs) or ESTs to the regions of the genome they are transcribed from (Ogasawara and Morishita, 2003; Mott, 1997; Gelfand et al., 1996). Alignment of draft sequences, where one or both of the sequences being aligned is split into a set of unordered contigs is another such example (Sundararajan et al., 2004; Bray et al., 2003; Delcher et al., 2002). Finding rearrangements between genomes (Abouelhoda and Ohlebusch, 2003; Brudno et al., 2003c) or finding differences between two assemblies of a genome (Lippert et al., 2004) are also closely connected to the fragment chaining problem. However, by far the most important application is linked to the alignment of large scale genomic sequences.

Global alignments are valuable tools for comparing close related genomes, and with some modifications that shall be discussed in the next chapter, for divergent genomes too. (Wilbur and Lipman, 1983; Sobel and Martinez, 1986) were the first to come up with the idea of fragmenting sequences in matching substrings, initially of fixed length, in order to speed up sequence alignment. Nowadays, the vast majority of tools for genome comparison (Hohl et al., 2002; Delcher et al., 2002; Brudno et al., 2003b; Brudno et al., 2003c; Bray et al., 2003; Darling et al., 2004; Morgenstern et al., 1998; Treangen and Messeguer, 2006) start by computing highly similar substrings, *i.e. fragments*, between the set of sequences. These similar substrings may have been identified by using several techniques and they can be *exact matches*, *approximate matches* or *local similarities*. For details on such genome comparison methods see Chapter 2 and Chapter 3.

The difficulty for the fragment chaining problem comes from the fact that fragments in the initial set may overlap or even be included one within another. Thus, once many such fragments have been found, the question is how one should select a subset among them that covers *best* the given sequences. Generally, in computational biology *best* is a difficult notion to define, as one does not have straightforward criteria for choosing such a subset. Nevertheless in practice, one picks up a model that does allow an efficient algorithm for selecting a best subset under that model. In such model, fragments have associated weights that account for their goodness, distances between fragments are being penalized, *i.e. gap costs* defined on page 97, Section 4.5, and all of these values are combined in order to obtain a weight of a subset of fragments. Another detail that can make the chaining problem more or less complicated is the number of sequences that are being compared. Therefore, in the next chapter we shall discuss separately the one-, two-, and multi-dimensional cases.

Moreover, several versions of the chaining problem exist but we shall give particular attention to the original version of the chaining problem. The original problem is known under the name of *collinear chaining* and it consists of selecting a best weighted subset of fragments that are collinear and do not overlap on any of the compared sequences. The subset of fragments computed in this way is what we usually call a *chain*. All the other versions of the chaining problem are generalizations of the

original one, usually by relaxing the collinearity constraint and, by abuse of language, the subset of fragments they compute is still called a chain.

The straight-forward method for the collinear fragment chaining problem is a special case of the classic optimum-path algorithm for directed acyclic graphs (Gusfield, 1997, chap. 13), which works for any definition of weights and gap costs. In this formulation, a quadratic time and space, dynamic programming algorithm can be easily applied by using a recurrence with respect to the partial order between fragments, see (Abouelhoda and Ohlebusch, 2005) for an outline of this algorithm and (Morgenstern, 2002) for a quadratic time solution needing linear space.

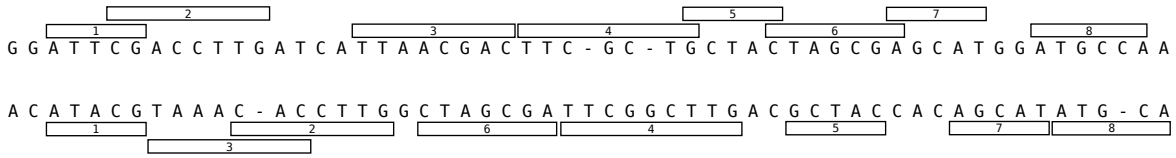
In order to speed up this algorithm, multiple formulations and solutions for this problem have been proposed in the literature. The general idea behind these solutions is *sparse dynamic programming* (Eppstein et al., 1992a). They make use of the partial order between fragments, employ weights that do not depend on independent positions and partition the search space into several regions allowing an efficient solution.

In Chapter 4, we start by giving a detailed description and precise formulation for the collinear chaining problem. Second, we present an efficient solution in this formulation. Next, we give an insight of other existing formulations and solutions for this problem. We then describe two generalization of the original problem: the *"glocal" chaining* and the *chaining with rearrangements*. Finally, even though there have been twenty years of research on this subject, we shall see that an interesting question still remain untackled. In practice, one would like to select all of the relevant fragments but many of these fragments may be overlapping. Thus, one would need an algorithm that efficiently selects these relevant fragments. In Chapter 5 we deal with this problem precisely and detail our results recently published in (Uricaru et al., 2010).

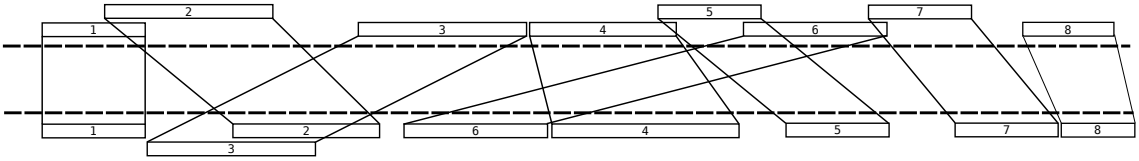
4.2 Preliminary notions for collinear chaining

Fragments are groups of highly similar substrings, one on each compared sequence. Let k be the number of compared genomic sequences. We may represent these k genomic sequences as k parallel lines, *i.e.* each sequence corresponds to one line. In this representation, a fragment is denoted by a set of k segments, one segment on each parallel line. Segments on a line are in a one to one correspondence with substrings on the genomic sequence. Moreover, the position of each segment on the line is given by the position of the corresponding substring in the genomic sequence. This representation of fragments as k-trapezoids is perfectly intuitive, as in reality when comparing genomic sequences, we visualize them as parallel lines. Now, if the extremities of the segments are connected, we obtain what we call a k-trapezoid. In Figure 4.1 a collection of two-dimensional fragments are represented as two-trapezoids.

Informally, we say that two k-trapezoids T_i and T_j are *strictly increasing* if the k segments composing T_i end before the k segments of T_j. This corresponds exactly to saying that the fragments they represent are *collinear*. Thus, a collinear set of fragments can be seen as a set of trapezoids that are *strictly increasing*.



(a)



(b)

Figure 4.1: A collection of two-dimensional fragments represented as trapezoids. (a) Eight fragments on two genomic sequences. The example is purely pedagogic and consists of fragments of minimum size 5, allowing gaps and mismatches. (b) The eight fragments above are now represented as eight two-trapezoids on two parallel lines. One may easily observe that a set of collinear fragments corresponds to a set of strictly increasing trapezoids, *i.e.* trapezoids having a disjoint intersection. In our example, fragments 1, 3, 4, 7, 8 are collinear, *i.e.* trapezoids 1, 3, 4, 7, 8 are strictly increasing, same as fragments 2, 6, 8.

Notation *Let T be a k-trapezoid that is defined by its k segments,* i.e. $T = (I^1, \ldots, I^k)$ *where I^α, with $1 \leq \alpha \leq k$, is the segment on line α. The left extremity of a segment I^α, respectively its right extremity, are denoted by $beg(I^\alpha)$, respectively by $end(I^\alpha)$. By extension, $beg(T) = (beg(I^1), \ldots, beg(I^k))$ and $end(T) = (end(I^1), \ldots, end(I^k))$ are points in $\mathbb{R}^k$ that correspond to the left, respectively to the right extremity of the trapezoid T. We remind the reader that if $x = (x_1, \ldots, x_k)$ and $y = (y_1, \ldots, y_k)$ are two points in $\mathbb{R}^k$, then $x < y$ is the classical dominance order between points in $\mathbb{R}^k$,* i.e. $\forall p\ 1 \leq p \leq k$, $x_p < y_p$.

Definition 4.1. *Let T_i and T_j be two k-trapezoids. We say that T_i and T_j are **strictly increasing** if $end(T_i) < beg(T_j)$. It is straight-forward that T_i and T_j are strictly increasing iff their intersection is disjoint,* i.e. $T_i \cap T_j = \emptyset$.

As collinear fragments are equivalent to strictly increasing trapezoids, meaning disjoint trapezoids, we obtain the following graph formulation of our problem. If we consider the collection of k-trapezoids as being a *trapezoid representation* of a *k-trapezoid graph*, we can model the collinear chaining problem as the *problem of finding a maximal independent set in a k-trapezoid graph.* Below we define *k-trapezoid graphs* as a special case of intersection graphs, see Definition 4.3, present an equivalent characterization of them, see Definition 4.5, and formulate the collinear chaining problem for this characterization in Section 4.3.

Definition 4.2 (Intersection graph)**.** *An **intersection graph** is an undirected graph*

formed from a family of sets S_i, with $i \leq n$, by creating n vertices, one vertex v_i for each set S_i. Two vertices v_i and v_j are connected by an edge whenever the corresponding two sets have a non empty intersection, i.e. *$\{v_i, v_j\}$ is an edge if and only if $S_i \cap S_j \neq \varnothing$.*

Several special classes of intersection graphs may be defined with respect to the types of sets that are used to form an intersection representation of them. For example, the intersection graph for a family of intervals is called an *interval graph* (Golumbic, 1980). In our case, the intersection graph for a family of k-trapezoids is called a *k-trapezoid graph* (Dagan et al., 1988). A formal definition of a k-trapezoid graph follows below.

Definition 4.3 (Trapezoid graph for a trapezoid representation)**.** *A graph is a **k-trapezoid graph** if there exists a collection of k-trapezoids between k parallel lines such that for each vertex v_i there is a trapezoid T_i and for every pair of vertices v_i, v_j there is an edge $\{v_i, v_j\}$ if and only if $T_i \cap T_j \neq \varnothing$. This family of trapezoids is called a **trapezoid representation** for the k-trapezoid graph. See Figure 4.2 for the two-trapezoid graph corresponding to the two-trapezoid representation in Figure 4.1.*

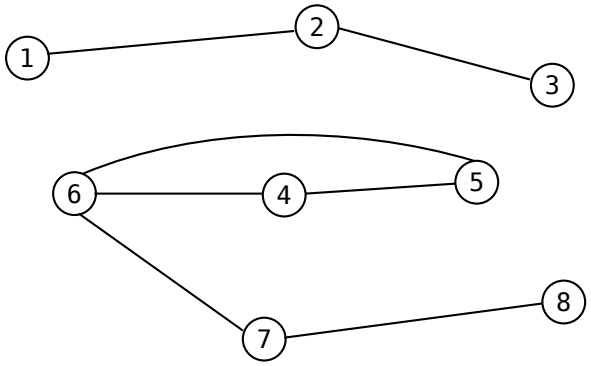


Figure 4.2: The trapezoid graph corresponding to the trapezoid representation in Figure 4.1, with each vertex corresponding to the trapezoid with the same number in Figure 4.1. Vertices are connected if corresponding trapezoids have a non empty intersection.

Next we introduce an equivalent characterization of trapezoid graphs: *the box representation*, (Felsner et al., 1995). We chose to give this second representation because it allows us to give an easier formulation for the collinear chaining problem. Instead of representing fragments with trapezoids, we now associate to each fragment a box. We shall see next how the two representations are connected.

For the trapezoid representation, genomic sequences were projected on parallel lines. In the case of the box representation, each genome is associated with one axis and *boxes* are axis parallel hyper-rectangles in $\mathbb{R}^k$. Thus, in the two-dimensional case, the input of the problem is represented as a set of rectangles drawn in the plane. For the multi-dimensional case, fragments may be represented as hyper-rectangles in a multi-dimensional space. As each side of a rectangle is parallel to one axis, it corresponds to a substring in the associated genomic sequence. Therefore, the length on a genome of

the fragment associated with a box is the projection of that box on the corresponding axis.

Notation *Let B be a box of $\mathbb{R}^k$ and let $\alpha \in \{1, 2, \ldots, k\}$ index the axis. The interval corresponding to the projection of B on axis α in the trapezoid representation is in fact the segment I^α. The **upper**, resp. **lower**, corner of B is denoted by $u(B)$, resp. $l(B)$. If T is the k-trapezoid corresponding to B in the trapezoid representation, then $l(B)$ is equivalent to $beg(T)$ and $u(B)$ to $end(T)$.*

Definition 4.4 (Box dominance order)**.** *Let B_i, B_j be two boxes of $\mathbb{R}^k$. We say that B_j **dominates** B_i, denoted $B_i \ll B_j$, if $l(B_j)$ dominates $u(B_i)$ in $\mathbb{R}^k$. If neither B_i dominates B_j, nor B_j dominates B_i, then B_i and B_j are **incomparable**.*

Property 4.1 (Transitivity of the dominance order)**.** *The dominance order between boxes is transitive.*

We then obtain a novel definition for the k-trapezoid graphs.

Definition 4.5 (Box representation of a trapezoid graph)**.** *A graph is a k-trapezoid graph if there exists a collection of boxes in a k-dimensional space such that for each vertex v_i there is a box B_i, and for every pair of vertices v_i, v_j there is an edge $\{v_i, v_j\}$ if and only if B_i and B_j are incomparable. We call this family of boxes a **box representation** for the k-trapezoid graph.*

In Figure 4.3 we illustrate the equivalent box representation for the trapezoid graph above. In order to understand that these two representations of k-trapezoid graphs are equivalent, we examine the two-dimensional case and observe that pairs of incomparable boxes are in one-to-one correspondence with trapezoid pairs linked by edges in the two-trapezoid graph. It is enough to map lower and upper segments in the trapezoid representation to the x-axis and the y-axis in the box representation, respectively. For a box B, its projections on the x-axis and the y-axis coincide with the lower and the upper segment, respectively of the two-trapezoid T. Thanks to the example in Figure 4.3, it can easily been seen that two trapezoids are disjoint exactly if the corresponding boxes are comparable. A similar reasoning remains valid for the multi-dimensional case.

Moreover, we may completely leave out the graph formulation for the collinear fragment chaining problem and switch between the k-trapezoid graph and the set of boxes equipped with the dominance order, *i.e. box order*. The dominance order between boxes, which is in fact a partial order between fragments, can be exploited in order to obtain more efficient algorithms than in the graph formulation by using computational geometry methods.

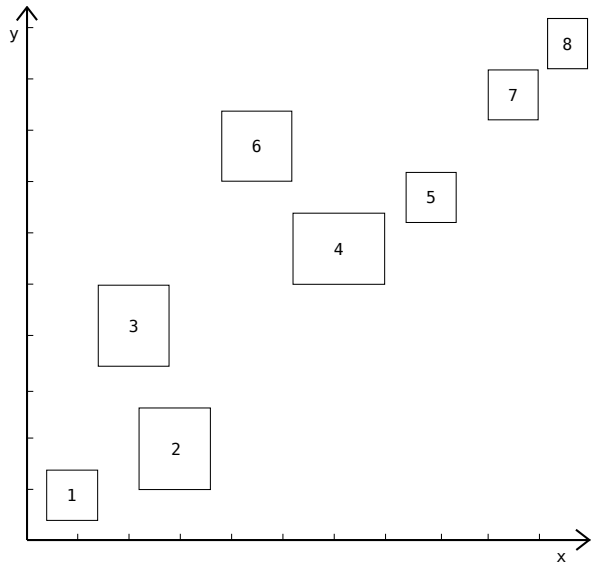


Figure 4.3: The equivalent box representation for the trapezoid graph in Figure 4.2 corresponding to the trapezoid representation in Figure 4.1. For example, we observe that $1 \ll 3$ meaning that the box 3 dominates the box 1, *i.e.* trapezoids 1 and 3 are strictly increasing, same as $3 \ll 4$, $4 \ll 7$, $7 \ll 8$.

4.3 The collinear chaining problem definition in a k-trapezoid graph and an equivalent box order

Let us consider a k-trapezoid graph $G = (V, E)$, whose trapezoid representation corresponds to a collection of fragments on k genomic sequences. On the vertices of G, we define a *weight function* $w : V \to \mathbb{R}$. Then, we may say that the k-trapezoid graph G is weighted. Moreover, the weight function can also be extended to the equivalent box order. Thanks to the bijection between boxes and vertices in the graph G, we may define $w(B_i) = w(v_i)$, where B_i is the box corresponding to the vertex v_i in G.

We remember that an *independent set* of G is a set of vertices, such that for every two vertices in this set, there is no edge connecting the two. As collinear fragments correspond to disjoint trapezoids, thus to unconnected vertices in the trapezoid graph, the Maximum Weighted Collinear Fragment Chain (MWCFC) problem is equivalent to the Maximum Weighted Independent Set (MWIS) problem in a trapezoid graph. Based on this equivalence, the weight of a collinear fragment chain can be defined as the sum of the weights of fragments in the chain, in the same way as the weight of an independent set of G is defined as the sum of the weights of its elements.

Definition 4.6. *The* Maximum Weighted Independent Set *(*MWIS*) problem in a graph G is to find the independent set of maximal weight among all independent sets in G.*

We remind the reader that a *chain* in an order is a set of mutually comparable elements of that order. Given the definition of comparable boxes in a box order, *i.e.*

Definition 4.4, (Felsner et al., 1995) observed that an independent set of G corresponds to a chain in the box order. Therefore, the relation between trapezoids and boxes makes that the collinear fragment chaining problem is also equivalent to the Maximum Weighted Chain (MWC) problem in the corresponding box order. From this point on, we shall leave aside graphs and work with the box order formulation.

Given an order, recall that a *maximal element* in a set is one with no other element dominating it. Each chain has exactly one maximal element. In the case of fragment chaining, box weights are arbitrary and they usually measure the "quality" of fragments (length, number of identities), see for example the weights chosen in Section 3.6.2. However, note that one should be able to obtain the weight of a box in constant time. This is an important observation to make for the complexity of the algorithms given in the next section. Moreover, the weight of a chain of the box order, is defined as the sum of the weights of its elements.

Definition 4.7 (Weight of a chain)**.** *Let $m \in \mathbb{N}$ and $\mathcal{B} := (B_1 \ll \ldots \ll B_m)$ be a chain of m boxes. The weight of $\mathcal{B}$ is $W(\mathcal{B}) := \sum_{i=1}^{m} w(B_i)$.*

Let $\mathcal{F}$ be a set of fragments and its corresponding box representation $\mathcal{B}' := \{B_2, \ldots B_{n-1}\}$. For convenience, we add two dummy boxes, B_1, B_n, such that for all $1 < i < n$: $B_1 \ll B_i \ll B_n$. Additionally, we set $w(B_1) = w(B_n) := 0$. Now, the input consists in $\mathcal{B} := \{B_1, \ldots, B_n\}$.

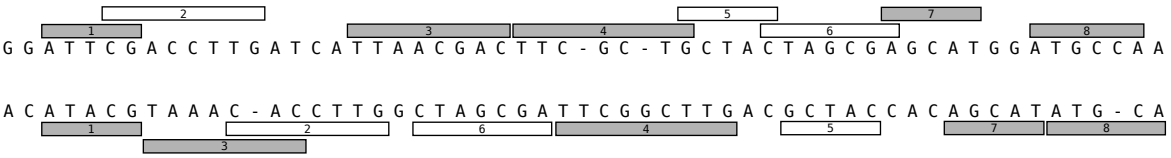
Definition 4.8 (Maximum Weighted Chain)**.** *Let $\mathcal{B} := \{B_1, \ldots, B_n\}$ a set of boxes. The* Maximum Weighted Chain *problem is to find in $\mathcal{B}$, according to the dominance order $\ll$, the **chain** C that starts with B_1 and ends in B_n and whose weight $W(C)$ is maximal.*

Property 4.2 (Best weighted collinear set of fragments)**.** *The best weighted chain in the box order corresponds to the best weighted collinear set of fragments in $\mathcal{F}$. See Figure 4.4 for an example of maximal weighted chain in a box order and its corresponding best weighted collinear set of fragments.*

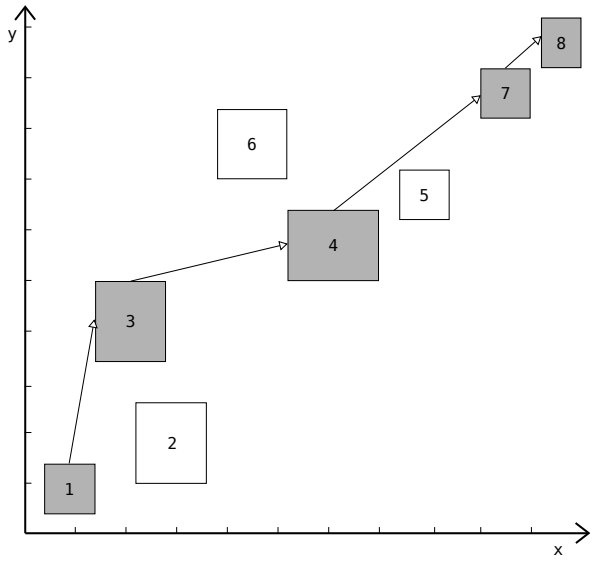
For any $1 \leq i \leq n$, let us denote by $\mathcal{C}_i$ the set of chains ending in B_i, and by $W(B_i)$ the weight of the maximal weighted chain in $\mathcal{C}_i$ (not to be confounded with $w(B_i)$). From now on, all the considered boxes belong to $\mathcal{B}$ unless otherwise specified.

4.4 Dynamic programming solution for the collinear fragment chaining in a box order

We begin this section by showing that the MWCFC problem can be solved by dynamic programming. For this we shall examine the equivalent MWC problem. Next we give a classical quadratic dynamic programming algorithm, before detailing in the next section a more efficient algorithm as introduced by (Felsner et al., 1995).



(a)



(b)

Figure 4.4: (a) A maximal weighted collinear set of fragments corresponding to the example in Figure 4.1 and (b) the respective maximal weighted chain in the equivalent box order in Figure 4.3.

Let us show that MWC can be solved by a dynamic programming algorithm. The definition of the weight of a chain suggests a recurrence equation to compute $W(B_i)$, with $W(B_1) = 0$ and for all $1 < i \leq n$:

$$W(B_i) = \max_{B_j:\, B_j \ll B_i} W(B_j) + w(B_i). \tag{4.1}$$

Obviously, this implies that for all $1 \leq j < n$ the value of $W(B_j)$ will be reused for computing $W(B_i)$ for every box B_i such that $B_j \ll B_i$. Thus, MWC consists of *overlapping subproblems*, which suits to the framework of dynamic programming (Cormen et al., 2001, chap. 15). However, it is correct to use Equation 4.1 only if our problem satisfies the condition of *optimal substructures* (Cormen et al., 2001, chap. 15). In Theorem 4.1, we easily show this is true.

Theorem 4.1 (Optimality of substructures for MWC)**.** *Let $m, i_1, \ldots, i_m$ be integers belonging to $[1, n]$, and let $D := (B_{i_1}, \ldots, B_{i_m})$ be an optimal weighted chain among the chains in $\mathcal{C}_{i_m}$. Thus, $D' := (B_{i_1}, \ldots, B_{i_{m-1}})$ is an optimal weighted chain among those in $\mathcal{C}_{i_{m-1}}$.*

Proof of Theorem 4.1 (Optimality of substructures). By hypothesis and Equation 4.1, one has

$$\begin{aligned} W(D) &= W(B_{i_m}) \\ &= w(B_{i_m}) + W(B_{i_{m-1}}) \\ &= w(B_{i_m}) + W(D'). \end{aligned}$$

We proceed by contradiction and assume that E' is a weighted chain ending in $B_{i_{m-1}}$ with $W(E') > W(D')$. Consider the chain $E := E' \cup \{B_{i_m}\}$. By the same reasoning as above, one has

$$W(E) = w(B_{i_m}) + W(E'),$$

and hence, $W(E) > W(D)$, contradicting the hypothesis that D is an optimal weighted chain ending in B_{i_m}. MWC problem satisfies the condition of *substructures' optimality.* □

The basic dynamic programming algorithm for this problem is given in Algorithm 1.

Algorithm 1: Dynamic Programming MWC in a Box Order

Data: $\mathcal{B}$ a set of n boxes
Result: W a vector of weights, with $W[B_n]$ the weight of the best chain in $\mathcal{B}$,
Pred a vector containing the previous boxes in the chain

```
1 begin
2     sort(B);
3     W[B_1] ⟵ 0;
4     Pred[B_1] ⟵ null;
5     foreach B_i ∈ B in order do
6         pred[B_i] ⟵ arg max_{B_j: B_j ≪ B_i} W[B_j] + w(B_i);
7         W[B_i] ⟵ W[pred[B_i]] + w(B_i);
8     traceback(Pred[B_n]);
9 end
```

The algorithm takes an initial set of n boxes, $\mathcal{B}$, and orders them according to Definition 4.4. For every box B_i taken in order, the algorithm finds the box B_j, which precedes B_i in the best weighted chain in $\mathcal{C}_i$, and stores it in Pred vector. The weight of the best chain ending in B_i, *i.e.* $W(B_i)$, is kept in a second vector $W[]$. The actual best chain of boxes, *i.e.* ending in B_n, is constructed by tracing Pred[] entries starting from Pred$[B_n]$.

As tracing back Pred[] can be done in linear time, the computation time of Algorithm 1 depends on the *sort*() procedure, and on the time needed to obtain Pred[] for a given box. Now, if we suppose that boxes are given already sorted, the computation time of the algorithm is $O(n \times g(n))$, where $O(g(n))$ is the time needed to compute Pred[]. Moreover one should observe that this time complexity is based on the assumption that the computation of $w(B)$ is negligible, *i.e.* it can be done in $O(1)$, like mentioned in the previous section.

If in the worst case, one needs to visit all the boxes in order to find the best predecessor for a given box, then $g(n)$ is in fact n. The whole running time complexity of the algorithm is in this case $O(n^2)$.

Theorem 4.2. *A dynamic programming algorithm solves the* Maximum Weighted Chain *problem in $O(n^2)$ time and $O(n)$ space.*

However, in the box order formulation by using complex data structures that allow less than $O(n)$ time for query and update procedures, one manages to improve this complexity and obtain a time bound inferior to $O(n^2)$, as shown in the next sections.

4.5 Common aspects among solutions for the collinear chaining

On an initial set containing n fragments, we have seen in Theorem 4.2 that the collinear chaining problem can be solved in $O(n^2)$ complexity with a classical dynamic programming method. Several more efficient solutions have been proposed in the literature starting from the early 90s. Even though they use different formulations, in order to speed up the running time of the algorithms, existing solutions for this problem, enumerated in Section 4.9, are essentially based on the following notions:

- **Sparse dynamic programming technique**. All of the efficient solutions that we present in this chapter use the dynamic programming technique in order to solve the chaining problem. As observed in (Eppstein et al., 1992a; Eppstein et al., 1992b), for this problem, only a sparse set of entries of the dynamic programming matrix matters for the optimization of the objective function. Therefore, methods take advantage of this sparsity in order to obtain algorithms with time complexities depending on the size of the sparse set and not on the size of the dynamic programming matrix.

- **Sweep line paradigm**. Several solutions employ the sweep-line technique, a key technique in computational geometry that consists in using an imaginary vertical line sweeping the entire plane and stopping on each point (Cormen et al., 2001, chap. 33). It deals with objects that either intersect or are in the immediate vicinity of the sweep-line. Once the sweep-line passes one of the objects, it usually places it into a dynamic data structure that takes advantage of the relationship between the objects, *i.e.* partial order between fragments in our case. The final results are available once the sweep-line has passed over all the objects. See Algorithm 2 in Section 4.6 for a detailed algorithm using this technique.

- **Division of the search space into several regions**. Based on the topological order between fragments, for every point, regions in the plane may be identified containing points that influence the computation of the current point.

- **Restrictions on gap costs**

  Gaps and gap costs The region between the extremities of two non-overlapping fragments on a genomic sequence is called a *gap*. If one associates a cost to placing a fragment after another in a chain, then this corresponds to the so-called *gap cost*.

  One may want to penalize long distances between fragments thus one may need to take gaps into account when computing the best weighted collinear chain. This adds an important difficulty to the problem of chaining. In fact, several existing solutions do not deal with gaps, like (Felsner et al., 1995) in Section 4.6. Solutions that choose to work with gaps use gap costs that essentially depend on the distance between fragments on the different sequences. In order for the solutions to stay efficient, gap costs must be defined in such a way that the particular symbols occurring in the substring between consecutive fragments must not count in the computation. This means that gap costs must be *symbol independent*, otherwise one needs to examine every such symbol thus increasing the overall time complexity.

Moreover, some of the solutions may require the use of special types of fragments, *i.e.* having identical lengths on all sequences, no gaps or no mismatches. They may also use data structures that take advantage of the fact that fragments coordinates are integers between 0 and the maximum size of the sequences, thus obtaining better time complexities.

4.6 An efficient solution for the collinear fragment chaining in the box order formulation

In this section, we detail an efficient solution for the MWCFC problem, as defined in Section 4.3. This solution was given in (Felsner et al., 1995). We shall see how, thanks to its geometrical properties, the box order formulation is more convenient than the trapezoid representation in designing efficient algorithms for the collinear chaining problem. Below, the algorithm is described in the general multi-dimensional case. However, examples, figures and additional explanations are given for the two-dimensional case.

Notation *We reuse the definitions and notation given in the previous sections:* $\mathcal{B}$ *a set of* n *boxes with* B_1 *and* B_n *such as for all* $1 < i < n$*:* $B_1 \ll B_i \ll B_n$*. For* B_i *a box in* $\mathcal{B}$ *we have* $l(B_i)$*,* $u(B_i)$ *the lower, respectively the upper corner of the box* B_i*,* $w(B_i)$ *the weight of the box,* $W(B_i)$ *the weight of the maximal weighted chain ending in* B_i*,* i.e. *among chains in* $\mathcal{C}_i$*. Moreover if* α *indexes an axis, for any point* $x \in \mathbb{R}^k$ *let* $P_\alpha(x)$ *denote its projection on axis* α*.*

Boxes are represented in a multi-dimensional space and the weight $W(B)$ for each box B is computed with the aid of a *sweep-line*. A vertical sweep line sweeps the boxes in the plane by increasing x-coordinates of their corners, stopping at the lower left and upper right corners of each box. To avoid visiting all possible predecessors as in the $O(n^2)$ dynamic programming algorithm when computing the best chain ending in B_i, a set, $\mathcal{A}$, of *active* boxes is maintained, with boxes that can compete for being the optimal predecessors in all chains. We note that in order to simplify the message of *Felsner et al.*, we renounced to their formulation with *markers* on upper corners and replaced the set of markers with a set of active boxes having the same utility as the markers.

Let $\mathcal{P}$ be an array containing the $2n$ points corresponding to $l()$ and $u()$ corners of the n boxes in $\mathcal{B}$. For each point they store to which box and to which corner it corresponds to. Points in $\mathcal{P}$ are ordered on their x-coordinates. *Felsner et al.* assume that the points in $\mathcal{P}$ have mutually different x-coordinates. Otherwise, they say slightly perturbing points having the same x-coordinates, such that lower corners have smaller x-coordinates than upper corners. Similar perturbations are done for the other coordinates.

In Algorithm 2, the main loop sweeps the points of $\mathcal{P}$ and processes in different manner lower (lines 7-9) and upper corners (lines 10-16). The weight of a chain ending in, say B_i, is computed when passing the lower corner of B_i. As mentioned above, only useful boxes are in $\mathcal{A}$, while the others are either not inserted, or deleted from $\mathcal{A}$ when they become useless. When sweeping the **lower corner** of B_i, one needs to examine boxes $B_j \in \mathcal{A}$ such that $u(B_j) < l(B_i)$, meaning $B_j \ll B_i$, in order to find the one having the best chain weight. In the two-dimensional case, due to the way $\mathcal{A}$ was built, this box is the first one "below and to the left" of the point $l(B_i)$, see Figure 4.5.

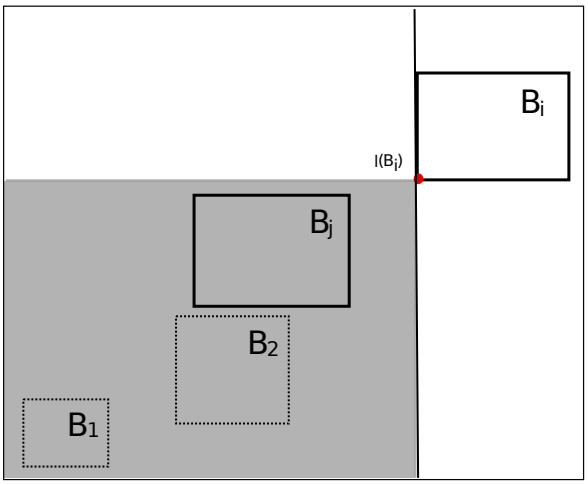


Figure 4.5: Two-dimensional example. The sweep line stops at the lower corner of the box B_i, *i.e.* $l(B_i)$. The best predecessor for B_i, B_j in this case, can be found immediately to the left and below $l(B_i)$.

When stopping at the **upper corner** of a box one needs to test whether it should be turned active and inserted in $\mathcal{A}$ (lines 12-13). The current box, B_i, is inserted only if there is no box that makes a better predecessor in $\mathcal{A}$. If it exists, this better predecessor is one of the boxes $B_j \in \mathcal{A}$ such that $u(B_j) < u(B_i)$. In the two-dimensional case, one finds this box "below and to the left" of the point $u(B_i)$, see Figure 4.6. If $W(B_j) > W(B_i)$, as the box B_j ends before the box B_i, then it necessarily makes a better predecessor for the boxes that have not been yet swept. In this case, B_i needs not be inserted in $\mathcal{A}$.

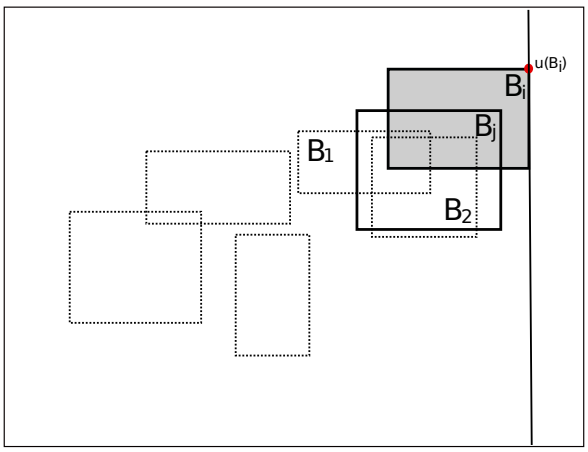


Figure 4.6: Two-dimensional example. The sweep line stops at the upper corner of the box B_i, *i.e.* $u(B_i)$. B_i is inserted in $\mathcal{A}$ if B_j, the first box below and to the left of $u(B_i)$, makes a worse predecessor than B_i, meaning that it has a smaller chain weight.

If B_i was inserted in $\mathcal{A}$, currently active boxes are investigated to determine if they are *less interesting* than B_i. An active box B_j is less interesting than B_i if $P_\alpha(u(B_j)) > P_\alpha(u(B_i))$, $\forall \alpha > 1$, *i.e.* in the two-dimensional case B_j is higher than B_i, and $W(B_j) < W(B_i)$, *i.e.* B_j is lighter than B_i. In this case, B_j is deleted from $\mathcal{A}$ (lines 14-16).

Felsner et al.. prove the correctness of their algorithm and set out the following theorem stating its time complexity when computing a collinear chain for an initial set of n fragments.

Theorem 4.3. *A maximal weighted chain of a box order in $\mathbb{R}^k$ containing n boxes, can be computed in $O(n \log^{k-1} n)$ time and $O(n \log^{k-2} n)$ space. This implies a $O(n \log n)$ time complexity and linear space in the two-dimensional case.*

The time complexity of the algorithm above depends on the time required by the *query* (searching for a box in $\mathcal{A}$), *insert* (inserting a box in $\mathcal{A}$) and *delete* (deleting a box from $\mathcal{A}$) operations. It can be easily observed that the upper bound of the number of times these operations are being carried out is $4n$. In fact, for each point in $\mathcal{P}$ a *find* operation is being performed, *i.e.* $2n$ times. Moreover, for each upper point, the corresponding box is added or not to $\mathcal{A}$, *i.e.* n boxes thus maximum n *insert*

Algorithm 2: Efficient MWC ($\mathcal{P}$)

Data: $\mathcal{B}$ a set of n boxes, $\mathcal{P}$ an array with the $2n$ box corners
Result: W a vector of weights, with $W[B_n]$ the weight of the best chain in $\mathcal{B}$, Pred a vector containing the previous boxes in the chain

```
1  begin
2      sort_on_x_coordinate(P);
3      A ⟵ B_1;
4      W[B_1] ⟵ 0;
5      Pred[B_1] ⟵ null;
6      foreach p ∈ P in ascending order on x-coordinate do
7          if p is a lower corner (i.e. ∃B_i : p = l(B_i)) then
               Pred[B_i] ⟵ argmax_{B_j∈A, u(B_j)<l(B_i)} (W[B_j]);
8
9              W[B_i] ⟵ W[Pred[B_i]] + w(B_i);
10         else                /* p is an upper corner, i.e.  ∃B_i : p = u(B_i) */
               B ⟵ argmax_{B_j∈A, u(B_j)<u(B_i)} (W[B_j]);
11
12             if W[B_i] > W[B] then
13                 A ⟵ A ∪ {B_i};
14                 foreach B_k ∈ A with P_α(u(B_k)) > P_α(u(B_i)), ∀α > 1 do
15                     if W[B_k] < W[B_i] then
16                         A ⟵ A \ {B_k};
17     traceback(Pred[B_n]);
18 end
```

operations. As boxes are inserted if ever, at maximum once, they can also be deleted at maximum once from $\mathcal{A}$, *i.e.* maximum n *delete* operations.

Now the time complexity depends on the characteristics of the data structure storing $\mathcal{A}$. In the two-dimensional case, boxes in $\mathcal{A}$ may be kept at the leaves of a *balanced binary tree* ordered according to the y-coordinate of their upper corners. If each node in the search tree points to the leaf corresponding to some box having maximal $W()$ among all leaves in the subtree rooted by this node, then finding a box having maximal $W()$ among boxes dominated by a box B, *i.e. find* operation, can be done in $O(\log n)$. Balancing the binary tree, inserting a new box, or deleting an existing one from the tree, can also be carried out in $O(\log n)$ time. This gives a total time complexity of $O(n \log n)$ as announced in Theorem 4.3. If storing active boxes from $\mathcal{A}$ in a binary tree and the $2n$ points from $\mathcal{P}$ in an array, linear space is used, *i.e.* $O(n)$.

For the multi-dimensional case, *i.e.* $k > 2$, a $(k-1)$-dimensional range tree is needed (De Berg et al., 2008, chap. 5). If the $k-1$ coordinates are noted from 1 to d, then the d dimensional range tree is built as follows. Boxes are represented by the leaves of a binary tree ordered according to the d-coordinate of their upper corners. If

$d = 1$ then every node points to the leaf corresponding to the box that has maximal $W()$ among all leaves in the subtree rooted by this node. If $d > 1$ the node points to a $(d-1)$-dimensional range tree with respect to the first $d-1$ coordinates for the boxes in the subtree rooted by this node. Finding a box having maximal $W()$ among boxes dominated by a box B can be done in $O(\log^d n)$ if the d dimensional range tree is balanced. The insertion and the deletion can be done in the same time. If at one of the levels the range tree becomes unbalanced, then it needs to be balanced and this can also be done in $O(\log^d n)$. Thus, the total time complexity of the algorithm is $O(n \log^{k-1} n)$. For the space complexity, as a box can be contained in at most $\log n$ tree levels, the total amount of space needed is $O(n \log^{k-2} n)$.

Observation 4.1. *Several observations on the solution presented above can be made:*

- *Felsner et al. do not take* ***gaps*** *into account when computing chain weights. In the following sections we shall mention several other solutions that accomodate gaps.*
- *The algorithm time complexity is due to the use of a set of active boxes,* i.e. *a subset of the initial set of boxes that may compete for being predecessors, thus to the fact that only a sparse set of boxes are being inspected at each step. Even though they do not mention it, we observe that the Felsner et al. algorithm is based on the* ***sparse dynamic programming technique****, similar to the other solutions presented below.*
- *In the computation of the time complexity, the average number of boxes in the set of active boxes is not taken into account, and it is upper bounded to the number of fragments. In practice, however, this number is significantly smaller than the total number of boxes. Thus, refining the time complexity remains an open problem.*
- *This solution can be easily adapted in order to solve the* ***one-dimensional*** *case,* i.e. *on a single sequence, see Section 4.7 for more details.*

4.7 The one-dimensional case of collinear chaining in the box order formulation

In computational biology, the one dimensional case of the collinear chaining problem, *i.e.* on one genomic sequence, arises in a simple version of the gene assembly problem, namely selecting a non-overlapping subset of exon candidates while ignoring intron candidates (Gusfield, 1997, chap. 18). However this approach does not work extremely well and more sophisticated approaches are being used in practice. It is the two-dimensional and its generalization to the multi-dimensional case that have the most interesting applications as previously enumerated.

First, remember the representations with trapezoids and then with boxes, and the definition for the multi-dimensional collinear chaining given in Sections 4.2 and 4.3.

For the one-dimensional case of the collinear chaining problem, fragments are in fact intervals on a line where each interval has an associated weight, usually the length of the interval. The simplest version of this problem consists in selecting a subset of non-overlapping intervals whose weights sum to the largest number possibly, see Figure 4.7.

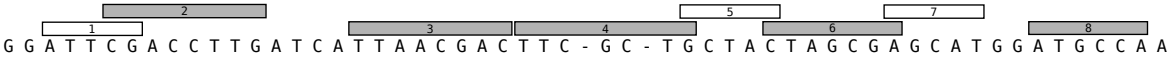


Figure 4.7: One-dimensional fragments represented as intervals on a genomic sequence and the maximal weighted subset of nonoverlapping intervals among them.

Similar to the multi-dimensional problem that corresponds to finding a maximal weighted independent set in a trapezoid graph, the one-dimensional case may be formulated as the problem of finding a *maximal weighted independent set in an interval graph*. Unfortunately, in this formulation, we do not obtain the most efficient algorithm for this problem.

Now if one uses the box representation in Section 4.3 and the definition of a chain in a box order, see Definition 4.8, one may adapt the algorithm of *Felsner et al.* and obtain an efficient solution.

It is straightforward that in the one-dimensional case, boxes consist of points in R^2, *i.e.* $l(B) = u(B)$ for each box $B \in \mathcal{B}$. Thus, in this case only one action is taken for a box, *i.e.* when the sweep line passes the corresponding point. In the same way as in the original algorithm, one may use a balanced binary tree containing at maximum n nodes. The upper bound of the number of times *find*, *insert* and *delete* operations are carried out is $3n$, and as each operation needs $O(\log n)$ time, the total time complexity is $O(n \log n)$.

4.8 Another formulation for the two-dimensional collinear chaining case: LCS based

K.P. Vo stumbled upon a problem intimately related to the collinear chaining, in a completely different field than computational biology. He was in fact working on a library for the cathode-ray tube (CTR) screen update to be distributed with System V Unix systems. The screen update algorithm must reduce the screen disturbances by matching screen lines, while giving preference to closely aligned matches. In this purpose, he defined a weighted extension of the *LCS problem* (Gusfield, 1997, chap. 12) involving a weight function that combines the lengths of the common substrings and the distances between them. The quadratic algorithm for this generalization of the LCS problem, known as the *Heaviest Common Subsequence (HCS) problem*, is detailed in (Vo, 1986).

Several years later, (Jacobson and Vo, 1992) proposed a faster solution derived from the algorithm for the LCS problem presented in (Apostolico and Guerra, 1987), which runs in $O(n \log n)$. Below, we detail their formulation, explain the relation with the

collinear chaining problem and give a short insight of their solution, based on the notion of *dominant matches* (Hirschberg, 1977).

Notation *Let $S = s_1 s_2 \dots s_t$ be a string over some alphabet $\mathcal{A}$. The ith symbol of S is denoted by s_i.*

Definition 4.9. *A string S' is called a **subsequence** of S if there is a sequence of integers $i_1 < i_2 < \dots < i_l$, such that S' is equal to $s_{i_1} s_{i_2} \dots s_{i_l}$.*

Definition 4.10. *By $S_{i\dots j}$ we denote a **substring**, meaning a contiguous subsequence of S, consisting of symbols from the symbol at position i to the symbol at position j. Similarly, the prefix of length j of the string S is represented by $S_{1\dots j}$.*

Definition 4.11. *A **common subsequence** between two strings S_1 and S_2, over some alphabet $\mathcal{A}$, is a subsequence that appears both in S_1 and in S_2. The **longest common subsequence problem** corresponds to finding a common subsequence whose length is maximal.*

We say that (i, j) is a **match** between S_1 and S_2 if the two symbols S_{1_i} and S_{2_j} are identical.

Definition 4.12. *A match (i, j) is considered to be **dominant** if every LCS of $S_{1_{1\dots i}}$ and $S_{2_{1\dots j}}$ ends at positions i in S_1, respectively j in S_2. Moreover, we say that a dominant match (i, j) is k-**dominant**, with $k \geq 1$, if the length of the LCS between prefixes $S_{1_{1\dots i}}$ and $S_{2_{1\dots j}}$ is k.*

Now, an LCS of S_1 and S_2 can be built by using only dominant matches. In fact, the length of an LCS corresponds to maximum value of k such that there exists a k-dominant match.

Let (i, j) and (i', j') be two k-dominant matches. If $i \geq i'$ and $j \geq j'$, then it is said that (i', j') is a better k-dominant match than (i, j), since any $(k + 1)$-dominant match that can be obtained from (i, j), can also be obtained from (i', j'). Thus, in order to recover the LCS we need to consider only dominant matches that are minimal under the vector ordering. Based on this observation, *Apostolico & Guerra* propose a dynamic programming algorithm for the LCS problem in $O(n \log n)$, where n is the maximum length among the lengths of S_1 and S_2.

Compared to the LCS problem, for the HCS problem we need to define weights on pairs of symbols. Given two strings S_1 and S_2, the weight of a common subsequence $S = s_1 s_2 \dots s_l$ is defined as $W(S) = \sum_{p=1}^{l} f(i_p, j_p, s_p)$, where (i_p, j_p) is a match for S_1 and S_2 and f is a weight function. The weight function f allows one to define weights that, besides depending on the symbols involved, may depend on the positions of the symbols in the two strings.

Definition 4.13. *The **heaviest common subsequence problem** (HCS) corresponds to finding a common subsequence of S_1 and S_2, S, having the biggest weight $W(S)$.*

(Jacobson and Vo, 1992) showed how to generalize the (Apostolico and Guerra, 1987) algorithm initially introduced for the LCS problem, in order to deal with the HCS problem. They obtained an algorithm running in $O(n \log n)$, which, in the case of constant weights, reduces to the (Apostolico and Guerra, 1987) LCS algorithm.

Now, if we return to the collinear chaining problem for two genomic sequences sequences, we can transform it into an HCS problem. If we interpret fragments as symbols composing a novel alphabet, we may define two new sequences, S_1 and S_2, each consisting of all fragments in the order in which they appear on the first and on the second genomic sequence respectively. However, directly transforming fragments into symbols can only be done in the particular case where fragments do not overlap. In general, fragments must be pre-processed so that they meet the previous property. In Figure 4.8, we take the example introduced in Section 4.2, trim the eventual overlaps between fragments, and build the S_1 and S_2 sequences. The HCS algorithm is then employed on S_1 and S_2, by using a weight function f that depends on the level of similarity of a fragment and on the cost for adding this fragment to the rest of the chain. However, in practice, trimming fragments is not an easy task, and we were not able to find any work on this subject, except some vague references.

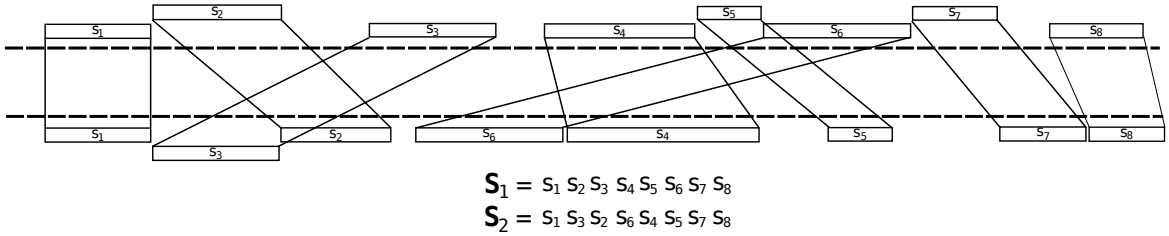


Figure 4.8: Eight fragments on two genomic sequences, each fragment corresponding to a symbol in S_1 and S_2. Compared to the example in Figure 4.1, in this case overlapping fragments have been trimmed . The HCS algorithm on S_1 and S_2 with a proper weight function returns a maximal weighted chain of fragments on the initial genomic sequences.

4.9 A glossary of solutions for the collinear chaining

In this section we enumerate several other efficient solutions for the collinear chaining problem, sorted on their year of publication. Some of these solutions deal with the multi-dimensional case, while others have been conceived for the two-dimensional case only and do not generalize readily to the multi-dimensional case. Obviously, the solutions for the multi-dimensional case remain valid in two-dimensions.

Two-dimensional solutions

- **LCS based formulation** The CRT screen update problem first led *K.P. Vo* in 1986 and finally *Jacobson & Vo* in 1992 to look into a weighted extension

for the LCS problem involving a weight function that combines the lengths of the common substrings and the distances between them. They generalize the LCS problem in a *Heaviest Common Subsequence (HCS) problem* and propose an algorithm that is a derivation of (Apostolico and Guerra, 1987) algorithm initially given for the LCS problem. Their algorithm is based on the idea of *dominant matches* and runs in $O(n \log n)$ for the two-dimensional case, (Jacobson and Vo, 1992). For more details see Section 4.8 above.

- **Two-track interval graph formulation.** (Joseph et al., 1992) propose a formulation for the two-dimensional fragment chaining problem based on a special case of the two-interval graphs, called the *two-track interval graphs*. Compared to common interval graphs, for the two-track interval graphs, intervals are placed on two parallel lines, *i.e.* two-track interval graphs are in fact synonym to two-trapezoid graphs seen in Section 4.2. Based on this formulation, they define the notion of *increasing independent set* and give an algorithmic solution to the collinear chaining problem that consists in finding an increasing independent set with maximal total weight. By employing the *sweep-line technique* they obtain an algorithm running in $O(n \log n)$. Even though their formulation does not generalize for the multi-dimensional case, the solution they propose is extremely similar to the solution explained in Section 4.6. Moreover, same as (Felsner et al., 1995), they do not take gaps into account in the process of weight computation.

- **Sparse dynamic programming approach.** Probably the most important advancement for the collinear chaining problem in the two-dimensional case is done in (Eppstein et al., 1992a; Eppstein et al., 1992b). In fact, previous solutions were already based upon the principle of *sparse dynamic programming* without explicitly naming it, but *Eppstein et al.* thoroughly studied this aspect. As already mentioned in Section 4.5, they explain that even though the recurrence for this problem is defined on a number of points that is quadratic in the input size, only a sparse set needs to be taken into account for the result. For this, they work with a geometrical representation of fragments, similar to the boxes representation in Section 4.2 and define the notion of *range*. By *range*, they mean that for each point, a geometric region can be found in the dynamic programming matrix, containing all the points that could have their value influenced by the current point.

  By considering fragments in order of increasing starting points on the first sequence, they need only to perform a one-dimensional search to find the appropriate range area for chaining each fragment. Specifically, this may be accomplished in $O(\log n)$ by storing range areas sorted by their main diagonal number in a balanced binary tree. Similar to the previous solution, they obtain a total running time of $O(n \log n)$, while managing to incorporate gaps when computing chain weights. Moreover, they manage to speed up this algorithm by taking into account the fact that the coordinates of points are integers between 0 and the maximum sequence length. Thanks to this observation, they use special data

structures, like the priority queue of Johnson (Johnson, 1982), and obtain an algorithm running in $O(n \log \log n)$.

Multi-dimensional solutions

- **Space division using kd-trees.** (Zhang et al., 1994) use higher dimensional computational geometry, namely *kd-trees*, in order to chain multiple alignment blocks. To avoid considering every predecessor of a given block, the idea is to cover the initial hyper-rectangle with smaller hyper-rectangles having sizes that do not exceed the so called "bucket size". As under some conditions certain hyper-rectangles do not need to be examined, thanks to this partition of the search space they manage to speed up the processing. Each node in the kd-tree corresponds to a hyper-rectangle. The root of the tree corresponds to the initial hyper-rectangle and the leaves constitute a pairwise disjoint decomposition of the initial hyper-rectangle. However, *Zhang et al.* are not capable of giving a theoretical analysis of their method's time complexity. In practice, for the two-dimensional case, experiments indicate that their method is far better than $O(n^2)$, although not as good as $O(n \log n)$. In fact, the running time behaviour seems highly sensitive to parameters, *i.e.* the bucket size, and data dependent.

- **Maximal weighted independent set in a k-trapezoid graph formulation.** (Felsner et al., 1995) work with the maximal weighted independent set in a k-trapezoidal graph and obtain an algorithm having $O(n\log^{k-1}n)$ time complexity. See the details on this solution in Section 4.6.

- In parallel with *Felsner et al.*, (Myers and Miller, 1995) propose in 1995 an algorithm for multiple chaining, incorporating gap costs, running in $O(n\log^k n)$. Their solution, though it does not use a graph formulation, is also based on the sweep-line paradigm and range trees. However, it is one log factor slower than *Felsner et al.* method or than previous methods for two-dimensional chaining. In 2005, (Abouelhoda and Ohlebusch, 2005) improve on their algorithm by a factor of $\log^2 n / \log \log n$ by efficiently using range trees and adapted priority queues (like Johnson priority queue). This is probably the solution with the best asymptotic worst time complexity for the multi-dimensional collinear chaining.

A general criticism on solutions described above would be that none of them thoroughly details the way that the weights of the fragments are being chosen. They usually name the *length of the fragment*, its *statistical significance* and the *number of identical symbols* in the fragment as possible choices for its weight. Moreover, besides restrictions on gap costs already discussed in Section 4.5, when dealing with gaps only two types of gap cost functions have been studied: gap costs simply defined as the distance between the end and start point of two fragments and the *sum-of-pairs* gap cost introduced in (Myers and Miller, 1995).

4.10 Chaining with rearrangements

Large scale alignment of genomic sequences involves new challenges compared with traditional sequence alignment. In this chapter we discuss one of these particular challenges, which makes collinear chaining insufficient: homologies do not necessarily appear in the same relative order in a set of homologous genomic sequences. In fact, genomes are known to undergo several types of large-scale evolutionary events: gene duplications, gene losses, horizontal transfers, reordering of genetic elements occurring by mechanisms such as repeated inversions or translocations. Genome comparison systems must account for all of these evolutionary phenomena to provide a complete picture of genetic differences among organisms. In this chapter, we do not insist on the different causes behind rearrangements, see Section 1.2 for a detailed discussion on this subject, but we deal with their effect on the disposition of fragments on the genomic sequences.

One needs to generalize the original definition of collinear chaining in order to deal with rearrangements, by removing the constraint on fragments being increasing on every sequence. If we keep the same notation with boxes representing fragments, see Section 4.3, we obtain the following formulation of the problem of chaining allowing for rearrangements.

First, we call two boxes being *conflicting* if their projections overlap on at least one of the axis; see Figure 4.9 for several configurations of conflicting boxes in the two-dimensional case. Such conflicting boxes correspond to fragments whose respective substrings on each genomic sequence are overlapping.

In Figure 4.9, boxes 2 and 3 are neither conflicting nor in a dominance relation; they are in fact the example of a common type of rearrangement event discussed in Section 1.2, *i.e. translocation*. Other rearrangement events that can be taken into account by chaining algorithms are *inversion* events, *i.e.* a part of one of the genomic sequences that has its direction reversed compared to the other sequences. One usually chooses a reference genomic sequence and considers inversions with respect to this sequence.

Definition 4.14. *We call a fragment corresponding to an inversion event an* ***inverted fragment****,* i.e. *on the reverse strand, in opposition to a* ***direct fragment****,* i.e. *on the direct strand.*

In the box formulation employed until this point, inversions cannot be directly observed. In fact, as it can be seen in Figure 4.10, boxes representing inverted fragments are identical to those representing direct fragments. Therefore, in order to deal with rearrangements, for a graphical purpose we need to upgrade the box formulation by adding *diagonals* inside boxes, *i.e.* the main diagonal for a direct fragment and the anti-diagonal for an inverted fragment.

Definition 4.15 (Chain with rearrangements)**.** *A set of boxes forms a chain with rearrangements if no two boxes in the set are mutually conflicting.*

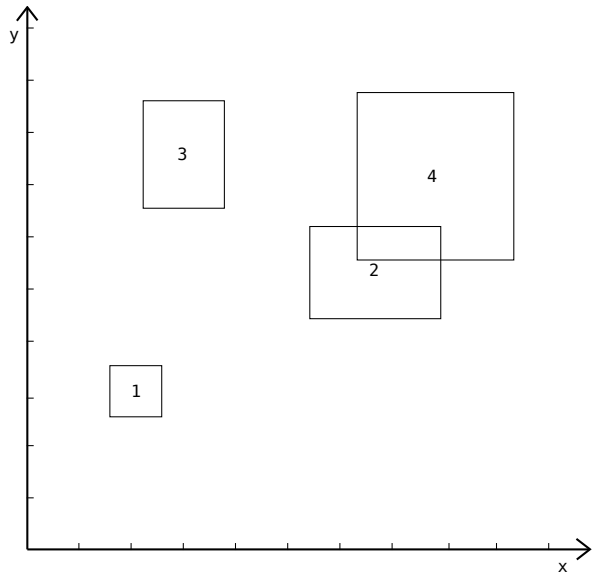


Figure 4.9: Several configurations of boxes in the plane for the two-dimensional case: boxes 1 and 3 are conflicting on x-axis, *i.e.* their projections on x-axis are overlapping, boxes 2 and 4 are conflicting on both axis, $1 \ll 2$ and $1 \ll 4$, boxes 2 and 3 are neither conflicting, nor in a dominance relation, and boxes 3 and 4 are conflicting as 3 is included in 4 on the y-axis.

Similar to a collinear chain in Definition 4.7, the weight of a chain with rearrangements is usually defined as the sum of the weights of boxes in the chain.

Definition 4.16. *The problem of chaining with rearrangements consists in finding a maximal weighted chain of boxes with rearrangements, see Figure 4.11.*

Similarly to the original collinear chaining, see Section 4.2, the chaining with rearrangements problem can also be formulated as a graph problem. If each node in the graph corresponds to a box in the set and there is an edge between two nodes if the corresponding boxes are conflicting, then the resulting graph is an intersection graph. Thus the current problem can be phrased as *the maximal independent set problem* in a corresponding intersection graph. Unfortunately, this problem is showed to be NP-complete in (Bafna et al., 1996) by reduction to the $3SAT$ problem. Therefore one needs to use approximation algorithms adapted to the particular case of genome comparison.

Besides the heuristics that one can use in order to find a satisfying solution to this problem, an intermediary formulation between the collinear chaining and the chaining with rearrangements, has been recently proposed in (Brudno et al., 2003c). They call this novel strategy *glocal chaining*. At this point, only a two-dimensional formulation and solution for the glocal chaining problem exists. Compared to the collinear chaining, the glocal chaining sorts fragments on one sequence only and puts no constraint on the second sequence, except for the fact that each two adjacent fragments must not overlap. Below we detail this type of approach and list the shortcomings

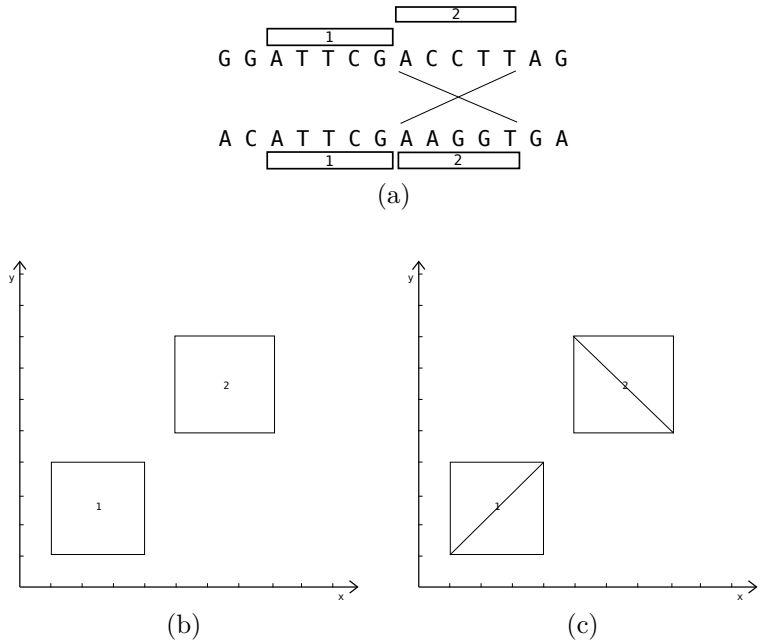


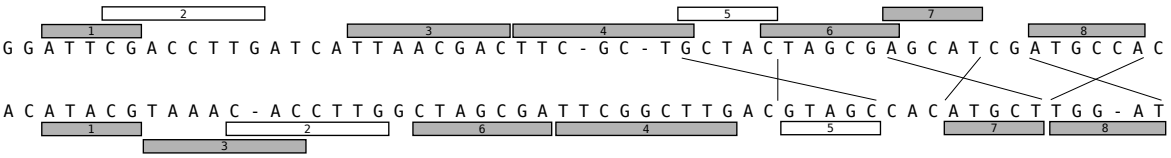
Figure 4.10: (a) Two fragments on two genomic sequences: fragment 1, a direct fragment and fragment 2, an inverted fragment. (b) Representation of the two fragments with boxes under the initial formulation; we observe that we cannot differentiate between an inverted and a direct fragment represented as boxes. (c) Fragments represented as boxes with the diagonal upgrade: main diagonal for fragment 1, anti-diagonal for fragment 2.

due to such method. Finally, we describe one of the heuristics used when dealing with rearrangements.

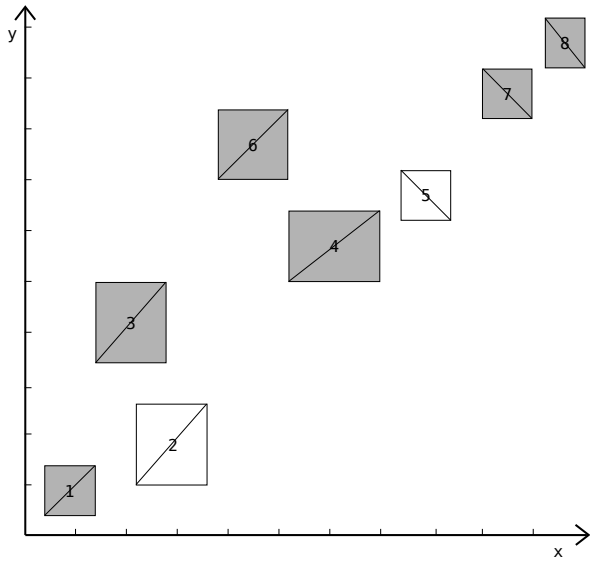
Once a chain of fragments with rearrangements has been computed, one may try to retrace a rearrangement scenario that transforms the reference sequence as to obtain the other sequences (Friedberg et al., 2008; Bourque and Tesler, 2008; Bérard et al., 2009).

4.11 Glocal chaining, a special type of chaining with rearrangements

Brudno et al. introduce for the first time in (Brudno et al., 2003c) the notion of *glocal chaining* in the two-dimensional case. In this section, we focus only on the chaining phase of the alignment tool Shuffle LAGAN (Brudno et al., 2003c); see the previous chapter for a global description of the method. The glocal strategy is a generalization of the collinear chaining that accommodates rearrangements. In (Brudno et al., 2003c), the glocal alignment is intuitively described as a series of operations that transform one sequence into the other. As each operation comes with a penalty and as the total *transformation distance* between the two sequences is given by the sum of these



(a)



(b)

Figure 4.11: (b) A maximal weighted chain of fragments with rearrangements corresponding to the example in Figure 4.1 and (b) the respective maximal weighted chain with rearrangements in the equivalent box order in Figure 4.3. Several modifications have been made to the original example in Figure 4.1 in order to accommodate some inversion cases.

penalties, the aim of a glocal chaining algorithm is to find the series of operations that generates the smallest total penalty. A similar idea can be found in (Varré et al., 1999), which proposed a distance metric between two DNA sequences that models various rearrangement events. Their algorithm could build a second sequence from an initially empty string using insertions and copying blocks from the first sequence, however it is incapable to handle simple edit operations.

Below we give the formulation for the glocal chaining problem using boxes as in the previous section, and give a short overview of the corresponding algorithm in this formulation. Their algorithm is a generalization of that given in (Eppstein et al., 1992a; Eppstein et al., 1992b) and shortly described in Section 4.9.

Notation *Remember the following notation from the previous sections reduced to the two-dimensional case. If* $\alpha \in \{1, 2\}$ *indexes an axis, for any point* $x \in \mathbb{R}^2$, $P_\alpha(x)$ *denotes its projection on axis* α. Let B be a box of $\mathbb{R}^2$, *i.e.* a rectangle in the plane. The interval corresponding to the projection of B on axis α is denoted $P_\alpha(B)$. The

upper, resp. *lower*, corner of B is denoted by $u(B)$, resp. $l(B)$.

Additionally, to identify the orientation of a box B in the two-dimensional case, we use the flag *strand*, *i.e.* $B.strand = +$ if the box corresponds to a direct fragment, or $B.strand = -$ if the box corresponds to an inverted fragment. Observe that even for an inverted box their lower and upper corners stay the same, *i.e.* the lower corner corresponds to the left lower corner of the box and the upper corner to its right upper corner. We may now define the 1-monotonic order between boxes in a glocal chain (in parallel with the box dominance order, from Definition 4.4, in a collinear chain).

Definition 4.17 (Box 1-monotonic order)**.** *Let B_i, B_j be two boxes of $\mathbb{R}^k$. We say that B_i and B_j are in 1-monotonic order, which we denote $B_i <_{1_{monot}} B_j$, if $P_1(u(B_i)) < P_1(l(B_j))$,* i.e. *$B_i$ and B_j are strictly increasing on the first axis, and non-overlapping on the second axis,* i.e. *(i) $P_2(u(B_i)) < P_2(l(B_j))$ or (ii) $P_2(u(B_j)) < P_2(l(B_i))$. Moreover, there is no constraint on their orientation: both boxes may be inverted, one inverted and one direct, or none of them inverted.*

Definition 4.18 (Glocal chain definition)**.** *A set of boxes $\mathcal{B} = \{B_1, B_2, \ldots, B_n\}$ composes a glocal chain if the set of boxes is 1-monotonic, meaning that for every pair of boxes B_i and B_{i+1} from $\mathcal{B}$, B_i and B_{i+1} are in a 1-monotonic relation,* i.e. *$B_i <_{1_{monot}} B_{i+1}$.*

Therefore, a set of boxes is 1-monotonic if it is strictly increasing on the first axis and has no overlaps between adjacent boxes on the second one, compared to a collinear chain of boxes that needs to be strictly increasing on both axis. A glocal chain of boxes corresponds to a glocal chain of fragments.

Observation 4.2. *Several observations on the glocal chain definition above can be made:*

- *Every collinear chain of boxes is also a glocal chain. The opposite affirmation is not true. In fact, the notion of glocal chain generalizes the notion of collinear chain.*
- *Two boxes corresponding to a translocation event are in a 1-monotonic relation and thus they can be taken together in a glocal chain.*
- *An essential observation, not mentioned in the paper of Shuffle LAGAN, is that the glocal chaining algorithm does not avoid overlaps between boxes in different LCBs on the second axis. More important even is that it allows overlaps without regulating their sizes and without taking them into account in the computation of weights. From what we know, boxes may even confound each other on the second axis and unfortunately, this raises a delicate question on the pertinence of the glocal strategy.*

Next, we present some notation given in (Brudno et al., 2003c) that we shall use in the rest of the section. When chaining a box B_i with a box B_j several configurations

may appear that can be denoted by three bits, one for the orientation of B_i, one for the position of B_i compared to B_j on the second axis, and a third one for the orientation of B_j. For example $(+++)$ codes for $B_i.strand = +$, $P_2(u(B_i)) < P_2(l(B_j))$ and $B_j.strand = +$, while $(---)$ for $B_i.strand = -$, $P_2(u(B_j)) < P_2(l(B_i))$, $B_j.strand = -$. There are such eight combinations corresponding to eight possible configurations between the box B_i and the box B_j. The original algorithm in (Eppstein et al., 1992a) deals with the special case where a direct fragment is always connected to another direct fragment, preceding it, *i.e.* $(+++)$.

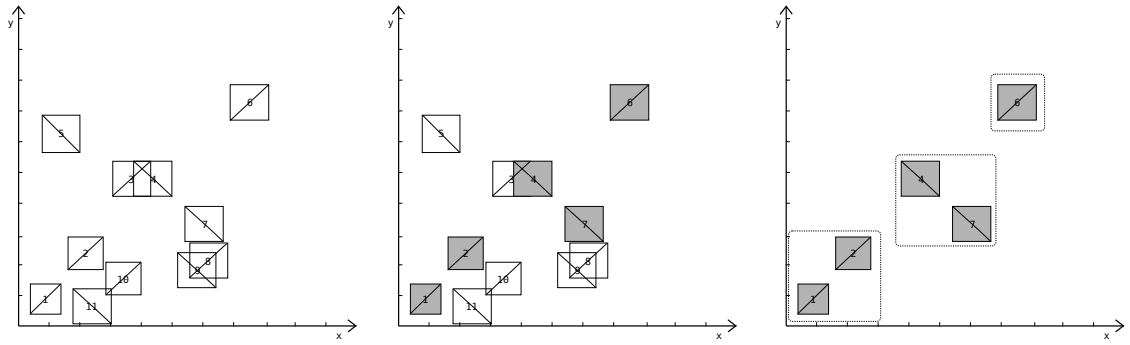
In order to compute an *optimal glocal chain*, penalties need to be fixed for chaining a box to an already formed subchain. Four different penalties are used: the regular gap penalty ($+++$ and $---$ cases), inversion penalty ($++-$ and $--+$ cases), translocated inversion penalty ($+--$ and $-++$ cases) and translocation penalty ($+-+$ and $-+-$). The sum of these penalties computed for a chain gives the total penalty of the glocal chain, corresponding to the total edit distance between the two sequences. The difficulty is finding good parameters for the various penalties, as there is no sound mathematical basis for this.

For each box B having a certain orientation, the idea is to look for the optimal collinear subchain with which to join B, in each of the four possible combinations described above. This is done in $O(\log n)$ time using balanced binary trees as in *Eppstein et al.*. The algorithm may be seen as running for each box four parallel versions of *Eppstein et al.* algorithm, one for each configuration, and taking the minimum over the four cases to compute a unique glocal chain. This gives a total running time of $O(n \log n)$. Once the 1-monotonic chain has been computed, it is partitioned in *locally collinear blocks* (LCBs), *i.e.* maximal subchains of boxes corresponding to homologous regions among the two sequences having a total order on the second axis; see a thorough definition of LCBs in the following section. In Figure 4.12 we give the graphical overview of the algorithm as presented in (Brudno et al., 2003c).

Once the 1-monotonic chain is found, it can be used to classify various types of rearrangements on it. In fact, one may determine whether most of one of the sequences is inverted, or may may identify local inversions, translocations and duplications on the second sequence. In fact, as one of the major drawbacks of the Shuffle LAGAN algorithm is its lack of symmetry in the sequence order, it misses duplications on the first sequence. Moreover, the scoring scheme is quite simplistic and it is likely to obtain better glocal chains with a more sophisticated scheme taking into account more operations and even combinations of operations. As mentioned above, the fact that boxes may overlap on the second axis is not at all discussed in the paper. Finally, an extension of the glocal alignment to multiple genomic sequences has not yet been suggested.

4.12 Heuristic for chaining with rearrangements

Chaining with rearrangements is a NP-complete problem, thus tools usually use heuristics approaches. Such heuristic for multiple chaining with rearrangements was



(a) (b) (c)

Figure 4.12: Overview of Shuffle LAGAN algorithm. (a) Fragments generated between the two sequences represented as boxes. (b) The optimal 1-monotonic chain (composed of dashed boxes) is found. (c) The maximal subchains of boxes, *i.e.* LCBs (dashed boxes inside rectangles); while boxes inside LCBs are not conflicting, boxes in different LCBs may be overlapping on the second axis, *e.g.* boxes 2, 7.

used in Mauve software (Darling et al., 2004) and is described below. Beside Mauve solution, other methods implementing variations on the problem of chaining with rearrangements are usually based on clustering approaches, see (Treangen and Messeguer, 2006; Pevzner and Tesler, 2003; Pevzner and Tesler, 2003; Calabrese et al., 2003). The main idea is to cluster fragments within some gap distance and then, to remove spurious clusters based on a basic length criterion or on more complex statistical criteria.

Starting from an initial set of fragments, Mauve chaining method identifies a minimum partition in regions of local homology called *locally collinear blocks* (LCBs), *i.e.* collinear subchains of fragments. In fact, each LCB is a homologous region of sequence shared by all of the genomic sequences under study, similar to the blocks composing the 1-monotonic chain in Shuffle LAGAN. In the general case, however, an LCB may be composed of sequence regions shared only by a subset of genomes. The collinear blocks selection is an intermediary step of Mauve global alignment method; the method as a whole was described in Section 2.5. In order to avoid spurious homologous regions, the locally collinear blocks are required to meet a user-specified minimum weight criteria, where the weight of an LCB provides a quality measure for the homologous region. The choice of the value for this minimum weight is empirical and it becomes a tradeoff between specificity and sensitivity.

Next we thoroughly define an LCB and explain the anchor selection algorithm meant to compute a minimum partitioning of the initial set of fragments in locally collinear blocks, which corresponds to a chain with rearrangements according to Definition 4.15.

Definition 4.19 (Locally Collinear Block (LCB))**.** *A collection of k-dimensional boxes $\{B_1, B_2, \ldots, B_{|lcb|}\}$ is said to form an LCB if it satisfies a total ordering property such that $\forall axis\alpha \in \{1, 2, \ldots, k\}$ (i) $P_\alpha(u(B_i)) < P_\alpha(l(B_{i+1}))$ if boxes in the LCB have a*

Function 3: `Partition into LCBs(`$\mathcal{B}$`)`

Data: $\mathcal{B}$ a set of n boxes
Result: CB a set of collinear blocks that partitions $\mathcal{B}$

begin
  $CB \longleftarrow \{cb_1\}$;
  foreach $B \in \mathcal{B}$ **do**
    $B.LCB \longleftarrow 1$;
  $\mathcal{B} \longleftarrow$ Sort on $P_1(l(B_i))(\mathcal{B})$;
  $label \longleftarrow 1$;
  foreach $B \in \mathcal{B}$ *in ascending order of their* $P_1(l(B))$ **do**
    $B.label \longleftarrow label$;
    $label \longleftarrow label + 1$;
  foreach $\alpha \in \{2, \ldots, k\}$ **do**
    $\mathcal{B} \longleftarrow$ `Sort on` $P_\alpha(l(B_i))$`(`$\mathcal{B}$`,` α`)`;
    $aux \longleftarrow 0$;
    $Seen\_LCBs \longleftarrow \emptyset$;
    foreach $B_i \in \mathcal{B}$ *in ascending order of their* $P_\alpha(l(B))$ *except the last* **do**
      $Seen\_LCBs \longleftarrow Seen\_LCBs \cup B_i.LCB$;
      if `Breakpoint on` α`(`B_i`,` B_{i+1}`)` *and* ($B_i.LCB = B_{i+1}.LCB$ *or* $B_{i+1}.LCB = aux$ *or* $B_{i+1}.LCB \in Seen\_LCBs$) **then**
        $CB \longleftarrow CB \cup cb_{|CB|+1}$;
        $aux \longleftarrow B_{i+1}.LCB$;
        $B_{i+1}.LCB \longleftarrow |CB|$;
      else
        if $B_{i+1}.LCB = aux$ & `Not(Breakpoint on` α`(`B_i`,` B_{i+1}`))` **then**
          $B_{i+1}.LCB \longleftarrow B_i.LCB$;
  foreach $B \in \mathcal{B}$ **do**
    $cb_{B.LCB} \longleftarrow cb_{B.LCB} \cup B$;
  return CB;
end

direct orientation on axis α *or (ii)* $P_\alpha(u(B_{i+1})) < P_\alpha(l(B_i))$ *if boxes have a reverse orientation on axis* α*, for* $\forall i \in \{1, 2, \ldots, |lcb| - 1\}$.

Let us denote the initial set of boxes by $\mathcal{B}$ and the weight of a box $B \in \mathcal{B}$ by $w(B)$. In (Darling et al., 2004), boxes are hyper-cubes, meaning that they have the same projection length on every axis (squares in the two-dimensional case), thus they consider $w(B)$ as being the length of such a projection, *i.e.* $P_\alpha(u(B)) - P_\alpha(l(B))$. The weight of an LCB, noted as $w(cb)$, is defined as the sum of the weights of the boxes composing it. Moreover, they consider a minimum weight threshold for a collinear block, *i.e.* $MinimumWeight$.

Algorithm 4: Greedy breakpoint elimination algorithm

Data: $\mathcal{B}$ a set of n boxes
Result: CB a set of collinear blocks that partitions $\mathcal{B}$

begin
  $z \longleftarrow MinimumWeight - 1$;
  while $z < MinimumWeight$ **do**
    $CB \longleftarrow$ `Partition into LCBs(`$\mathcal{B}$`)`;
    foreach $cb \in CB$ **do**
      $w(cb) \longleftarrow \sum_{B \in cb} w(B)$;
    $z \longleftarrow \min_{cb \in CB} w(cb)$;
    if $z < MinimumWeight$ **then**
      foreach $cb \in CB$ *with* $w(cb) = z$ **do**
        $\mathcal{B} \longleftarrow \mathcal{B} \setminus \bigcup_{B \in cb} B$;
end

Darling et al. mention that the minimum partitioning of an initial set of boxes into LCBs can be found by using breakpoint analysis (Sankoff and Blanchette, 1998). Mauve implements a greedy algorithm that removes repeatedly low weight LCBs from $\mathcal{B}$, based on this minimum weight criteria. See Algorithm 4 and Figure 4.13, extracted from (Darling et al., 2004), for a general overview of their algorithm. Observe that the algorithm needs non-overlapping fragments, thus overlapping fragments are trimmed before launching the partitioning algorithm. However the trimming procedure is not described in the paper.

In Function 3 we given an insight of their procedure for a minimum partitioning into collinear blocks. First, Mauve orders boxes in $\mathcal{B}$ on $P_1(l(B_i))$ and assigns monotonically increasing labels between 1 and $|\mathcal{B}|$ to each box, corresponding to its index in the ordering. Let us denote by $B.label$ the label of B. After this initial ordering all boxes are a part of a unique LCB indexed with 1. We refer to the LCB to which a box belongs to with $B.LCB$, and for now every box B has $B.LCB = 1$.

Next, they repeatedly re-order $\mathcal{B}$ on $P_\alpha(l(B_i))$ for $\alpha \in \{2, \ldots, k\}$. After each re-ordering, they search the $\mathcal{B}$ set for *breakpoints*, see Section 2.1.3 for a biological meaning of breakpoint regions. In this formulation a breakpoint exists on axis α between two boxes B_i and B_{i+1} if $B_i.label + 1 \neq B_{i+1}.label$ and both B_i and B_{i+1} are in direct orientation on axis α, or $B_i.label - 1 \neq B_{i+1}.label$ and both B_i and B_{i+1} are inverted. Moreover, they also call a breakpoint if B_i is in a different orientation than B_{i+1} on axis α. For each two consecutive boxes B_i and B_{i+1} in this ordering on axis α, if they are part of the same LCB, *i.e.* $B_i.LCB = B_{i+1}.LCB$, and there is a breakpoint between them, then the LCB is divided in two and an index for a novel LCB is created.

In the end, boxes are re-ordered on $B.label$ and the LCB partition is traced back based on LCB indexes of boxes, *i.e.* all boxes having the same LCB index are contained

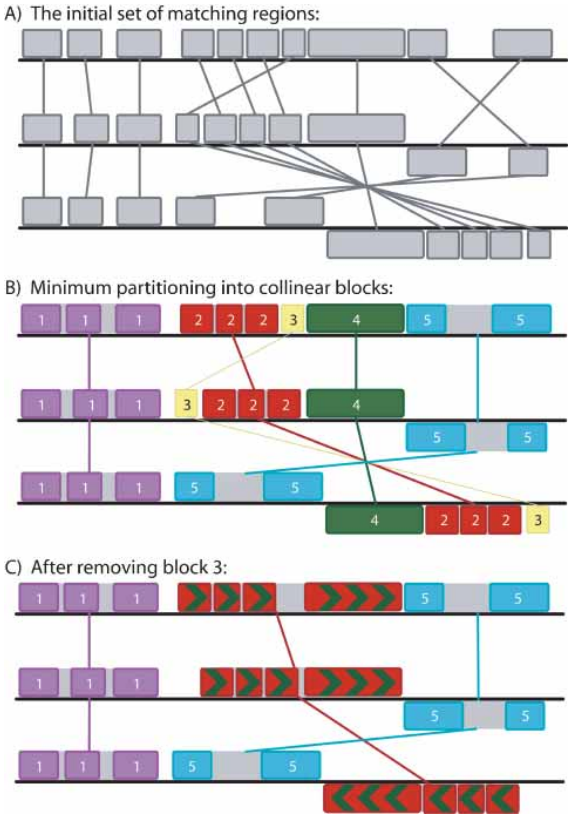


Figure 4.13: A pictorial representation of greedy breakpoint elimination strategy for three genomes. (A) The algorithm begins with the initial set of fragments represented as connected blocks. Blocks below a genome sequence line are inverted relative to the reference sequence. (B) The fragments are partitioned into a minimum set of collinear blocks. Each sequence of identically coloured blocks represents a collinear set of homologous regions. One connecting line is drawn per collinear block. Block 3 (yellow) has a low weight relative to other collinear blocks. (C) As low-weight collinear blocks are removed, adjacent collinear blocks coalesce into a single block, potentially eliminating one or more breakpoints. Figure extracted from (Darling et al., 2004).

in the same LCB. See Function 3 for the complete algorithm.

Concerning the time complexities of Algorithm 4 and in particular of Function 3, *Darling et al.* make no statement. However, we suspect that for Function 3 the time complexity is upper bounded by the repeated orderings of the set of boxes, *i.e.* $O(kn \log n)$. Therefore, we should obtain a total time complexity of $O(kn^2 \log n)$.

5 Personal contribution. Novel problem of chaining with proportional overlaps

Allowing overlaps between fragments in a chain has proven to be mandatory. Therefore we propose a generalization of the collinear chaining problem by allowing proportional overlaps between adjacent fragments in the chain. We formalize the newly introduced problem, give an efficient algorithm for it, prove its correctness and discuss its complexities.

Contents

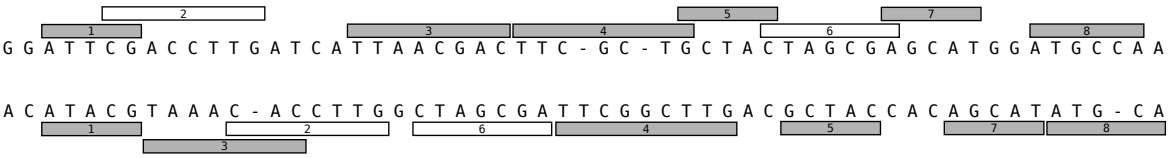
5.1 Introduction to collinear chaining allowing overlaps

When chaining genomic fragments, overlaps of variable sizes between fragments, from extremely small to extremely large, are very frequent. Such overlaps are due to randomness, to methodological reasons during the fragment computation step, mostly responsible for short overlaps, and also to biological reasons like *tandem repeats*, which generate longer overlaps. More details on these reasons behind overlaps were given in Section 3.6.3. The presence of overlaps prevents building the largest possible chain of fragments, as conflicting fragments cannot be taken together in the chain. In Section 3.6.3 we discuss that in practice, we would like to allow some of these overlaps in order to avoid fragmented chains and thus maximize the total length of the chained fragments.

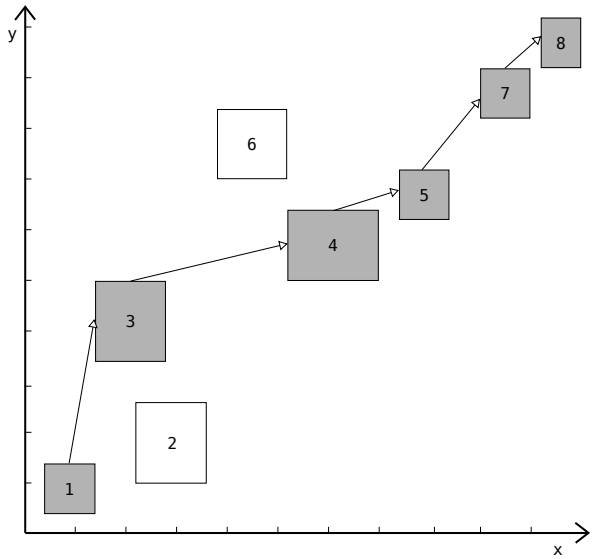
Here we focus on the solutions that chaining algorithms should implement in order to deal with overlaps. Unfortunately, there is not much work mentioned on this subject in the literature. Up to now, chaining methods dealt with overlaps in two different manners: whether (i) they completely interdicted them like Mauve and most of the methods in Section 4.9, or (ii) they allowed them inside the chain, without any restriction and no additional processing, like (Eppstein et al., 1992a; Pevzner and Tesler, 2003; Brudno et al., 2003c). Overlaps between adjacent fragments in the chain are usually prevented by defining dominance orders that do not allow overlaps. Other methods, like Mauve, completely trim overlaps between fragments before the chaining step; however, this pre-processing is usually not documented even though we consider it as being far from trivial. In the following sections we shall address the problem of collinear chaining with overlaps.

As we have seen in the previous chapter, chaining algorithms usually seek to maximize the total length of the chained fragments, whether the chain allows for rearrangements or not. In the case of collinear chaining, given the set of n shared genomic intervals, *i.e.* fragments, the MWC problem is solved in $O(n \log n)$ time by dynamic programming when overlaps between adjacent fragments are forbidden (Myers and Miller, 1995; Abouelhoda and Ohlebusch, 2005). The algorithms enumerated in Section 4.6 and in Section 4.9 can be extended to handle fixed length overlap between adjacent fragments but, as we shall see next, this is not sufficient to deal with the large differences in fragment lengths.

Small overlaps are often caused by equality over a few base pairs of fragment ends due to randomness, since the alphabet has only four letters. To handle such cases, a trivial solution would be to set a constant, large enough, maximal allowed overlap threshold. An $O(n \log n)$ time algorithm for the MWC with Fixed Length Overlap problem, whose chain weight function accounted for overlaps, was designed and used for mapping spliced RNAs on a genome (Shibuya and Kurochkin, 2003), but the fixed bound on overlaps remains a limitation. Another method that, based on their sayings, deals with overlaps can be found in (Delcher et al., 1999). They implement a variation of the LCS algorithm, described in Section 4.8. However, they give no details on the chaining algorithm, which seems to have a $O(n \log n)$ time complexity. Regarding overlaps they give no further details; they most probably put no constraint on the size



(a)



(b)

Figure 5.1: (a) A maximal weighted collinear set of fragments with overlaps, corresponding to the example in Figure 4.1 and (b) the respective maximal weighted chain in the equivalent box order in Figure 4.3. Observe that compared to the initial example, when allowing for overlaps we can take box 5 together with the other boxes in the maximal weighted chain, even it is conflicting with box 4 on the second axis.

of accepted overlaps or, in the best scenario, handle with fixed length overlaps as in (Shibuya and Kurochkin, 2003).

However, biological structures like tandem repeats (TR) that vary in number of copy units generate overlaps that are large relatively to the fragments involved. To illustrate this case, let u, v, w be words and assume the sequences of two genomes G_a, G_b are $G_a = uvvw$ and $G_b = uvvvw$, *i.e.* contain a variable TR of motif v. Then, uvv generates a local alignment between G_a and G_b, as well as vvw, but both fragments overlap over v in both G_a and G_b. Since v can be large, such cases cannot be circumvented with fixed length overlaps: only proportional overlaps can handle these. To raise the fixed length limitation, we formulate the MWC with Proportional Length Overlap problem (MWC-PLO) and exhibit the first collinear chaining algorithms allowing for overlaps that are proportional to the fragment lengths, and whose chain weight function accounts for overlaps.

In the next sections, we detail our results published in (Uricaru et al., 2010). We

define the *collinear chaining with proportional overlaps* based on a novel dominance order between boxes called *tolerance dominance order*. See Figure 5.1 for an example of a collinear chain with overlaps. We set the dynamic programming framework and detail the algorithm that solves it. Next, following (Felsner et al., 1995), we use the box representation of a trapezoid graph and adapt the sweep line paradigm to this problem, see Section 4.6, in order to obtain an efficient solution. We prove its correctness and discuss the complexities. The formulation and solution described in the following sections correspond to the two-dimensional case. Even though a multi-dimensional extension seems feasible, we did not tackle this problem for the time being and we leave it as future work.

5.2 A novel tolerance definition for the Maximal Weighted Chain problem in a box order

Notation *In this section we follow the same notation for a box order given in Chapter 4. We shall however extend this notation as to take overlaps into account. Let* $\alpha \in \{1, 2\}$ *index the axis, and for any point* $x \in \mathbb{R}^2$ *let* $P_\alpha(x)$ *denote its projection on axis* α*. Let* I *be an interval of* $\mathbb{R}$ *and* $\mathcal{I}$ *be a set of disjoint intervals of* $\mathbb{R}$*; we denote by* $|I|$ *the length of* I *and by* $|\mathcal{I}|$ *the sum of the lengths of intervals in* $\mathcal{I}$*. By extension, the interval corresponding to the projection of* B *on axis* α *is denoted* $P_\alpha(B)$*.*

To formulate a MWC with Proportional Length Overlap problem (MWC-PLO) in our framework, we need to redefine the dominance order, *i.e.* the original is given in Definition 4.4. The novel definition needs to accept overlaps that are proportional to the boxes' projection lengths, and the weight function has to truly measure the size of the chain on each genome. This requires that the chain weight counts only once a subinterval covered by several overlapping fragments.

Let $r \in [0, 1[$ represent the maximal allowed overlap ratio between any two boxes.

Definition 5.1 (r tolerant dominance order)**.** *Let* B_u *and* B_v *be two boxes.* B_v dominates B_u on axis α *in this* tolerant dominance order*, denoted by* $B_u \ll_{r,\alpha} B_v$*, iff*

$$P_\alpha(u(B_u)) - P_\alpha(l(B_v)) \leq r \min(|P_\alpha(B_u)|, |P_\alpha(B_v)|).$$

Now, we denote by $B_u \ll_r B_v$ *the fact that* B_v dominates B_u *if and only if for each* $\alpha \in \{1, 2\}$ $B_u \ll_{r,\alpha} B_v$*.*

A set of mutually comparable boxes with the order above is called a *chain* of boxes in the tolerance dominance order. All chains we refer to in the following of the chapter are such chains.

Next we show that the dominance between boxes implies the dominance between their upper, resp. lower, corners. Moreover, this tolerant dominance order is transitive.

Property 5.1. *Let* B_u, B_v *two boxes such that* $B_u \ll_r B_v$*. Then* $l(B_u) < l(B_v)$ *and* $u(B_u) < u(B_v)$*.*

Proof of Property 5.1 (dominance between upper, resp. lower corners). We prove by contradiction and first assume that $l(B_u) \geq l(B_v)$. We obtain on axis α

$$\begin{aligned} P_\alpha(u(B_u)) - P_\alpha(l(B_v)) &= P_\alpha(u(B_u)) - P_\alpha(l(B_u)) + (P_\alpha(l(B_u)) - P_\alpha(l(B_v))) \\ &= |P_\alpha(B_u)| + (P_\alpha(l(B_u)) - P_\alpha(l(B_v))) \\ &\geq \min(|P_\alpha(B_u)|, |P_\alpha(B_v)|) \\ &> r \min(|P_\alpha(B_u)|, |P_\alpha(B_v)|) \end{aligned}$$

Hence, as this contradicts Definition 5.1, we obtain $l(B_u) < l(B_v)$. Proving that $u(B_u) < u(B_v)$ can be done in a similar manner. □

Property 5.2. *The dominance order $\ll_r$ is transitive.*

Proof of Property 5.2 (transitivity of $\ll_r$). Let B_t, B_u, B_v be three boxes such that $B_t \ll_r B_u$ and $B_u \ll_r B_v$. We will show that $B_t \ll_r B_v$. Let $\alpha \in \{1, 2\}$. By hypothesis and from Property 5.1, we obtain both $l(B_t) < l(B_u) < l(B_v)$ and $u(B_t) < u(B_u) < u(B_v)$. From these we get both

$$P_\alpha(u(B_t)) - P_\alpha(l(B_v)) < P_\alpha(u(B_t)) - P_\alpha(l(B_u)) \leq r \min(|P_\alpha(B_t)|, |P_\alpha(B_u)|), \quad (5.1)$$

and

$$P_\alpha(u(B_t)) - P_\alpha(l(B_v)) < P_\alpha(u(B_u)) - P_\alpha(l(B_v)) \leq r \min(|P_\alpha(B_u)|, |P_\alpha(B_v)|). \quad (5.2)$$

When combined, these equations imply

$$\begin{aligned} P_\alpha(u(B_t)) - P_\alpha(l(B_v)) &\leq r \min(|P_\alpha(B_t)|, |P_\alpha(B_u)|, |P_\alpha(B_v)|) \\ &\leq r \min(|P_\alpha(B_t)|, |P_\alpha(B_v)|)), \end{aligned}$$

and hence $B_t \ll_r B_v$. □

From Property 5.1, one deduces the following corollary, which will help to compute efficiently the weight of overlapping boxes in a chain.

Corollary 5.1. *Let B_t, B_u, B_v be three boxes such that $B_t \ll_r B_u \ll_r B_v$. Then $(B_t \cap B_v) \subset (B_u \cap B_v)$.*

We define the *weight of a box* as the sum of lengths of its projections on all axis, and the *weight of a chain* of boxes in the tolerance dominance order as the sum of the projection lengths of all boxes on each axis, while counting only once subintervals covered by several boxes.

Definition 5.2 (Weight of a box, of a chain)**.** *Let B be a box and $\alpha \in [1,2]$. Its weight on axis α is $w_\alpha(B) := |P_\alpha(B)|$, and its weight is $w(B) := \sum_{\alpha=1}^{2} w_\alpha(B)$. Let $m \in \mathbb{N}$ and $C := (B_1 \ll_r \dots \ll_r B_m)$ be a chain of m boxes. The weight of C on axis α, denoted $W_\alpha(C)$, is*

$$W_\alpha(C) := \left| \bigcup_{i=1}^{m} P_\alpha(B_i) \right|,$$

while its weight *is $W(C) := \sum_{\alpha=1}^{2} W_\alpha(C)$.*

Note also that the weight of a box only depends on the endpoints of its projection on each axis, and hence, can be computed in constant time.

Clearly, it can be easily seen that

$$\begin{aligned} W_\alpha(C) &= w_\alpha(B_m) + \sum_{j=1}^{m-1} \left| P_\alpha(B_j) \setminus \bigcup_{l=j+1}^{m} P_\alpha(B_l) \right| \\ &= w_\alpha(B_m) + \sum_{j=1}^{m-1} |P_\alpha(B_j) \setminus P_\alpha(B_{j+1})| \quad \text{by Corollary 5.1.} \end{aligned} \tag{5.3}$$

The following easy property will also prove useful.

Property 5.3. *Let B_t, B_u two boxes such that $B_t \ll_r B_u$. Then*

- *$B_t \cap B_u$ is an, eventually empty, axis parallel rectangle of $\mathbb{R}^2$, and*
- *for $\alpha \in [1,2]$, $|P_\alpha(B_t) \setminus P_\alpha(B_u)| = |P_\alpha(B_t) \setminus P_\alpha(B_t \cap B_u)| = w_\alpha(B_t) - w_\alpha(B_t \cap B_u)$.*

Now, we can define the MWC-PLO problem. Let $\mathcal{B}' := \{B_2, \dots B_{n-1}\}$ be the set of input boxes. As in Section 4.3, for convenience, we add two dummy boxes, B_1, B_n, such that for all $1 < i < n$: $B_1 \ll_r B_i \ll_r B_n$. Additionally, we set $w(B_1) = w(B_n) := 0$. Now, the input consists in $\mathcal{B} := \{B_1, \dots, B_n\}$.

Definition 5.3 (MWC with Proportional Length Overlap)**.** *Let $r \in [0,1[$ and $\mathcal{B} := \{B_1, \dots, B_n\}$ a set of boxes. The* MWC with Proportional Length Overlap *problem is to find in $\mathcal{B}$, according to the dominance order $\ll_r$, the* chain *C that starts with B_1 and ends in B_n and whose weight $W(C)$ is maximal.*

Similar to Section 4.3, for any $1 \leq i \leq n$, let us denote by $\mathcal{C}_i$ the set of chains ending in B_i, and by $W(B_i)$ the weight of the maximal weighted chain in $\mathcal{C}_i$ (not to be confounded with $w(B_i)$). From now on, all the considered boxes belong to $\mathcal{B}$ unless otherwise specified.

5.3 Dynamic programming framework and solution for the collinear chaining with overlaps

In the same way as for MWC in Section 4.4, here we show that MWC-PLO can be solved by a dynamic programming algorithm. Equation 5.3 suggests a recurrence

equation to compute $W(B_i)$, with $W(B_1) = 0$ and for all $1 < i \leq n$:

$$W(B_i) = \max_{B_j:\, B_j \ll_r B_i} W(B_j) + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(B_j)| \,. \tag{5.4}$$

Obviously, this implies that for all $1 \leq j < n$ the value of $W(B_j)$ will be reused for computing $W(B_i)$ for every box B_i such that $B_j \ll_r B_i$. Thus, MWC-PLO consists of *overlapping subproblems*, which suits to the framework of dynamic programming (Cormen et al., 2001, chap. 15). However, it is correct to use Equation 5.4 only if our problem satisfies the condition of *optimal substructures* (Cormen et al., 2001, chap. 15). In Theorem 5.1, we show this is true.

Theorem 5.1 (Optimality of substructures for MWC-PLO)**.** *Let $m, i_1, \ldots, i_m$ be integers belonging to $[1, n]$, and let $D := (B_{i_1}, \ldots, B_{i_m})$ be an optimal weighted chain among the chains in $\mathcal{C}_{i_m}$. Thus, $D' := (B_{i_1}, \ldots, B_{i_{m-1}})$ is an optimal weighted chain among those in $\mathcal{C}_{i_{m-1}}$.*

Proof of Theorem 5.1 (Optimality of substructures). By hypothesis, Equation 5.3 and Property 5.3, one has

$$\begin{aligned} W(D) &= W(B_{i_m}) \\ &= w(B_{i_m}) + \sum_{j=i_1}^{i_{m-1}} \sum_{\alpha} |P_\alpha(B_j) \setminus P_\alpha(B_{j+1})| \\ &= w(B_{i_m}) - w(B_{i_m} \cap B_{i_{m-1}}) + w(B_{i_{m-1}}) + \sum_{j=i_1}^{i_{m-2}} \sum_{\alpha} |P_\alpha(B_j) \setminus P_\alpha(B_{j+1})| \\ &= w(B_{i_m}) - w(B_{i_m} \cap B_{i_{m-1}}) + W(D') \,. \end{aligned}$$

We proceed by contradiction and assume that E', rather than D', is an optimal weighted chain ending in $B_{i_{m-1}}$, *i.e.* $W(E') > W(D')$. Consider the chain $E := D' \cup \{B_{i_m}\}$. By the same reasoning as above, one has

$$W(E) = w(B_{i_m}) - w(B_{i_m} \cap B_{i_{m-1}}) + W(E'),$$

and hence, $W(E) > W(B_{i_m})$, contradicting the hypothesis that D is an optimal weighted chain ending in B_{i_m}. MWC-PLO satisfies the condition of *substructures' optimality.* □

The MWC with Proportional Length Overlap can thus be solved by a dynamic programming algorithm. The basic algorithm for this problems is similar to Algorithm 1 in Section 4.4 and can be described as in Algorithm 5.

Algorithm 5 uses two n-element arrays: $W[.]$ and Pred[.] to store for all $1 \leq i \leq n$ resp. the values of $W(B_i)$ and the predecessor of B_i in an optimal weighted chain ending in B_i. If searching for the best predecessor of B_i consists in examining all boxes B_j with $B_j \ll_r B_i$, then this algorithm takes $O(n^2)$ time and $O(n)$ space. In Section 5.4 we give another algorithm for MWC-PLO that is more efficient in practice.

Theorem 5.2. *A dynamic programming algorithm solves the* MWC with Proportional Length Overlap *problem in $O(n^2)$ time and $O(n)$ space.*

Algorithm 5: Dynamic Programming MWC-PLO in a Box Order

Data: $\mathcal{B}$ a set of n boxes
Result: W a vector of weights, with $W[B_n]$ the weight of the best chain in $\mathcal{B}$,
Pred a vector containing the previous boxes in the chain

1 **begin**
2 $sort(\mathcal{B})$;
3 $W[B_1] \longleftarrow 0$;
4 $\text{Pred}[B_1] \longleftarrow null$;
5 **foreach** $B_i \in \mathcal{B}$ *in order* **do**
6 $\text{Pred}[B_i] \longleftarrow \underset{B_j:\, B_j \ll_r B_i}{\arg\max} (W[B_j] + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(B_j)|)$;
7 $W[B_i] \longleftarrow W[\text{Pred}[B_i]] + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(\text{Pred}[B_i])|$;
8 $traceback(\text{Pred}[B_n])$;
9 **end**

5.4 An efficient solution for the collinear fragment chaining with overlaps in a box order

Following *Felsner et al.*, we give a sweep line algorithm in which a vertical line sweeps the boxes in the plane by increasing x-coordinates of their corners, stopping at the lower left and upper right corners of each box. To avoid visiting, as in Algorithm 5, all possible predecessors when computing the best chain ending in B_x, we maintain a set, $\mathcal{A}$, of *active* boxes that can compete for being the optimal predecessor in that chain. The idea is the same as in Section 4.6 but as predecessors can overlap B_x, this computation involves several steps, meaning that $W[B_x]$ and $\text{Pred}[B_x]$ can be updated several times before getting their final value. This is the main difference with Algorithm 5 and with *Felsner et al.*.

As in Section 4.6, let $\mathcal{P}$ be an array containing the $2n$ points corresponding to $l()$ and $u()$ corners of the n boxes in $\mathcal{B}$. Points in $\mathcal{P}$ are ordered on their x-coordinates. Compared to *Felsner et al.*, we allow points to have identical x-coordinates, and among these points having identical x-coordinates, lower corners are placed before upper corners. For each point, we store to which box and to which corner it corresponds to. In Algorithm 6, the main loop sweeps the points of $\mathcal{P}$ and processes in a different manner lower (lines 8-11) and upper corners (lines 12-24). We say a box B_x is *open* when the sweep line is located between $l(B_x)$ and $u(B_x)$ inclusive, *closed* when the line has passed $u(B_x)$, and *future* when it lies before $l(B_x)$. These states are exclusive of each other, and partition at each moment $\mathcal{B}$ in three disjoint sets (see Figure 5.2). All open boxes at each point are kept in a set $\mathcal{O}$ (lines 9, 13). The weight of a chain ending in, say B_i, and passing by a predecessor of B_i, B_x, can only be computed when B_x is closed (when $W[B_x]$ has reached its final value). If $P_1(u(B_x)) < P_1(l(B_i))$ then this can be done when stopping at $l(B_i)$ (lines 10-11), while if B_x overlaps B_i on

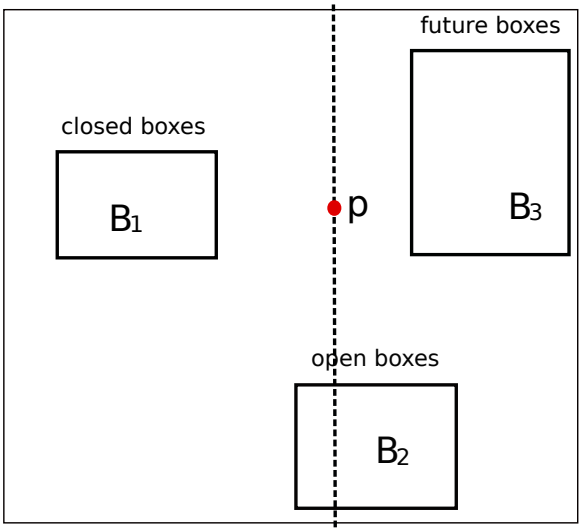


Figure 5.2: Example of boxes in each disjoint set forming a partition of $\mathcal{B}$, when sweeping a point p, *i.e.* open, closed and future boxes.

x-axis, then this is done when stopping at $u(B_x)$, and at the same time for all open boxes having B_x as predecessor (lines 14-18). These two cases partition the possible predecessors of B_i according to the location of their upper corners in two areas $A_b(B_i)$ and $A_o(B_i)$ (cf. Figure 5.3).

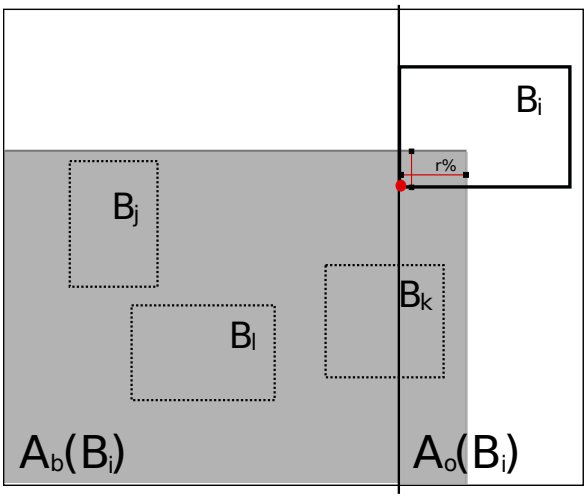


Figure 5.3: Partition of the search space of possible predecessors of B_i, according to the location of their upper corners, in two areas $A_b(B_i)$ and $A_o(B_i)$. $A_b(B_i)$ and $A_o(B_i)$ partition the rectangle delimited by a solid line: $A_b(B_i)$ is at left from the dashed line, and $A_o(B_i)$ at its right.

As above mentioned, we maintain in $\mathcal{A}$ the set of interesting predecessors for all future boxes. Boxes in $\mathcal{A}$ are *active* boxes. Hence, once closing a box (stopping at its upper corner), we test whether it should be turned active and inserted in $\mathcal{A}$ (lines

19-21). The current box, B_i, is inserted only if we cannot find a better predecessor in $\mathcal{A}$. Afterwards, if B_i has been added, currently active boxes are investigated to determine if they are less interesting than B_i, in which case they are deleted from $\mathcal{A}$ (lines 22-24). Active boxes are consulted when opening a box B_i, for computing the best chain ending in B_i with a predecessor in $A_b(B_i)$ (lines 10-11).

5.4.1 Correctness of the Algorithm

For $1 \leq i \leq n$, we show that $W[B_i]$ and $\text{Pred}[B_i]$ store the weight and the predecessor of B_i in a maximum weighted chain ending in B_i. First, several simple invariants emerge from Algorithm 6. I_1: At any point, the set $\mathcal{O}$ contains all open boxes. I_2: Both $W[B_i]$ and $\text{Pred}[B_i]$ store their final values once $u(B_i)$ has been processed, since they are not altered after that point. I_3: Hence, at any point all active boxes (*i.e.* boxes in $\mathcal{A}$), which are closed boxes, satisfy I_2. For conciseness, as $W[B_i]$ and $\text{Pred}[B_i]$ are computed jointly, from now on we deal only with $W[B_i]$. Since potential predecessors of B_i are partitioned in $A_b(B_i)$ (Figure 5.4) and $A_o(B_i)$ (Figure 5.5), we will prove two invariants: I_4: partial optimality over $A_b(B_i)$ at lower corners, and I_5: optimality at upper corners.

I_4: partial optimality over $A_b(B_i)$ at lower corners. We show that after processing $l(B_i)$, $W[B_i]$ stores the weight of a maximum weighted chain ending in B_i with predecessor in $A_b(B_i)$. Given line 10, this is equivalent to showing that no better chain ending in B_i passes through a potential predecessor that does not belong to $\mathcal{A}$ at that point, which we prove by contradiction. While processing $l(B_i)$, $\mathcal{A}$ contains a subset of boxes in $A_b(B_i)$, but obviously none from $A_o(B_i)$. Let B be a closed box of $\mathcal{B} \setminus \mathcal{A}$ such that $B \ll_r B_i$ and $w(B_i) - w(B \cap B_i) + W[B] > W[B_i]$, in other words, B makes a better predecessor for B_i than those in $\mathcal{A}$. From $B \ll_r B_i$, we get

$$P_2(u(B)) - P_2(l(B_i)) \leq r \min(|P_2(B)|, |P_2(B_i)|) \ . \tag{5.5}$$

As only two possibilities exist for B not belonging to $\mathcal{A}$, we distinguish two exclusive cases.

B was not turned active when sweeping $u(B)$ (lines 19-21). B did not satisfy the condition on line 20. Let $B' := \underset{B_j \in \mathcal{A}: u(B_j) < u(B)}{\arg\max} (W[B_j])$. Our hypothesis means that $u(B') < u(B)$ and

$$W[B] < W[B'] \tag{5.6}$$
$$|P_2(B)| \leq |P_2(B')| \ . \tag{5.7}$$

For B does not overlap B_i and $u(B') < u(B)$, we have B' does not overlap B_i on the x-axis. From $u(B') < u(B)$, we get $P_2(u(B')) < P_2(u(B))$; this with equations 5.5

Algorithm 6: MWC_Tolerance_Box_Order ($\mathcal{P}$)

Data: $r \in [0,1[$, $\mathcal{B}$ a set of n boxes, $\mathcal{P}$ an array with the $2n$ box corners
Result: W a vector of weights, with $W[B_n]$ the weight of the best chain in $\mathcal{B}$, Pred a vector containing the previous boxes in the chain

begin
  sort_on_x_coordinate($\mathcal{P}$);
  $\mathcal{A} \longleftarrow B_1$;
  $W[B_1] \longleftarrow 0$;
  $\text{Pred}[B_1] \longleftarrow null$;
  $\mathcal{O} \longleftarrow \emptyset$;
  foreach $p \in \mathcal{P}$ *in ascending order on x-coordinate* **do**
    if p *is a lower corner (*i.e. $\exists B_i : p = l(B_i)$*)* **then**
      $\mathcal{O} \longleftarrow \mathcal{O} \cup \{B_i\}$;
      $\text{Pred}[B_i] \longleftarrow \underset{B_j \ll_r B_i, B_j \in \mathcal{A}}{\arg\max} (W[B_j] + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(B_j)|)$;
      $W[B_i] \longleftarrow W[\text{Pred}[B_i]] + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(\text{Pred}[B_i])|$;
    else `/* p is an upper corner, i.e.` $\exists B_i : p = u(B_i)$ `*/`
      $\mathcal{O} \longleftarrow \mathcal{O} \setminus \{B_i\}$;
      foreach $B_k \in \mathcal{O}$ *with* $B_i \ll_r B_k$ **do**
        $w_k \longleftarrow W[B_i] + \sum_{\alpha=1}^{2} |P_\alpha(B_k) \setminus P_\alpha(B_i)|$;
        if $w_k > W[B_k]$ **then**
          $W[B_k] \longleftarrow w_k$;
          $\text{Pred}[B_k] \longleftarrow B_i$;
      $B \longleftarrow \underset{u(B_j) < u(B_i), B_j \in \mathcal{A}}{\arg\max} (W[B_j])$;
      if $W[B_i] \geq W[B]$ *or* $|P_2(B_i)| > |P_2(B)|$ **then**
        $\mathcal{A} \longleftarrow \mathcal{A} \cup \{B_i\}$;
        foreach $B_k \in \mathcal{A}$ *with* $P_2(u(B_k)) > P_2(u(B_i))$ **do**
          if $W[B_k] < W[B_i]$ *and* ($|P_2(B_k)| < |P_2(B_i)|$ *or* $P_2(l(B_k)) > P_2(u(B_i))$) **then**
            $\mathcal{A} \longleftarrow \mathcal{A} \setminus \{B_k\}$;
  traceback($\text{Pred}[B_n]$);
end

and 5.7 yields

$$\begin{aligned} P_2(u(B')) - P_2(l(B_i)) &< P_2(u(B)) - P_2(l(B_i)) \\ &\leq r \min(|P_2(B)|, |P_2(B_i)|) \\ &\leq r \min(|P_2(B')|, |P_2(B_i)|). \end{aligned} \tag{5.8}$$

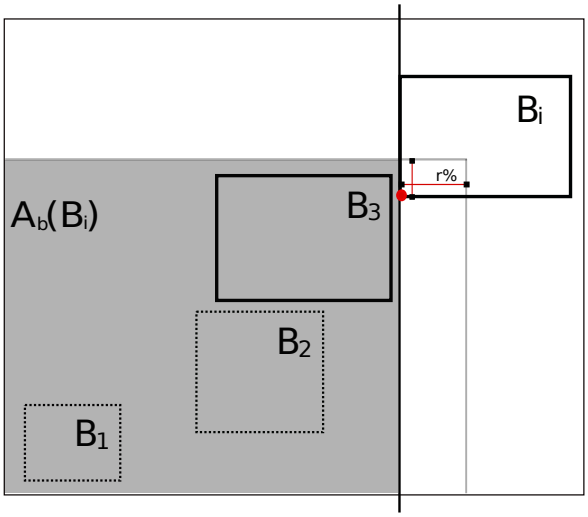


Figure 5.4: When the sweep line passes $l(B_i)$, Pred$[B_i]$ is a partial optimum on the set of possible predecessors of B_i lying in $A_b(B_i)$. In the example, B_3 is the best current predecessor of B_i.

Equation 5.8 and B' not overlapping B_i on the x-axis imply $B' \ll_r B_i$. Finally, from equations 5.6, 5.7, and $u(B') < u(B)$ we obtain:

$$W[B] + \sum_{\alpha=1}^{2}(w_\alpha(B_i) - w_\alpha(B_i \cap B)) \quad < \quad W[B'] + \sum_{\alpha=1}^{2}(w_\alpha(B_i) - w_\alpha(B_i \cap B')),$$

and thus B' makes a better predecessor for B_i than B, a contradiction.

B was inactivated when sweeping $u(B_k)$ for some box B_k ending before $l(B_i)$ (lines 22-24). The hypothesis means that B was deleted from $\mathcal{A}$ for it satisfied $P_2(u(B_k)) < P_2(u(B))$, $W[B] < W[B_k]$, and at least one of the conditions (a) $|P_2(B)| < |P_2(B_k)|$ or (b) $P_2(u(B_k)) < P_2(l(B))$.

a) As above (see Eq. 5.8), from Equation 5.6, from $|P_2(B)| < |P_2(B_k)|$, and $P_2(u(B_k)) < P_2(u(B))$, we get

$$P_2(u(B_k)) - P_2(l(B_i)) \quad < \quad r \min(|P_2(B_k)|, |P_2(B_i)|)\,. \tag{5.9}$$

Moreover, as B_k does not overlap B_i on the x-axis, we obtain $B_k \ll_r B_i$. As $P_2(u(B_k)) < P_2(u(B))$, B_k and B do not overlap B_i on x-axis, and $W[B] < W[B_k]$, we finally derive

$$W[B] + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(B)|) < W[B_k] + \sum_{\alpha=1}^{2} |P_\alpha(B_i) \setminus P_\alpha(B_k)|)\,. \tag{5.10}$$

b) By hypothesis, we know that $P_2(u(B_k)) < P_2(l(B)) < P_2(l(B_i))$, and since neither B_k nor B overlap B_i on the x-axis, we directly obtain $B_k \ll B_i$ ($B_i \cap B_k = \emptyset$). Thus, $W[B] < W[B_k]$ also implies Equation 5.10.

With either condition (a) or (b), B_k makes a better predecessor for B_i than B, a contradiction.

Finally, after processing $l(B_i)$, $W[B_i]$ stores the weight of a maximum weighted chain ending in B_i with predecessor in $A_b(B_i)$, which concludes the proof of I_4.

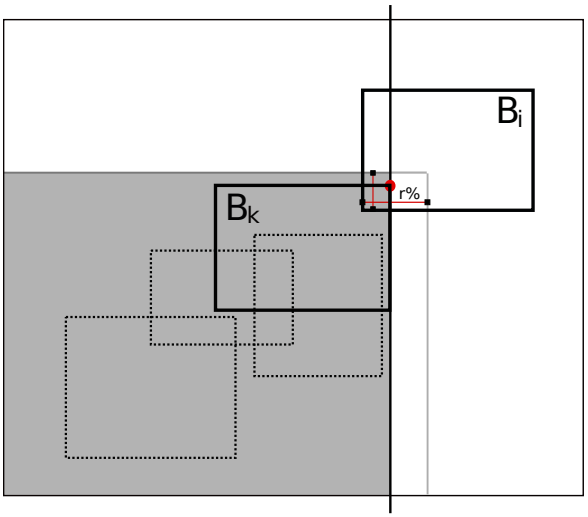


Figure 5.5: When the sweep line passes $u(B_k)$, Pred$[B_i]$ is a partial optimum on the set of possible predecessors of B_i from $A_b(B_i) \cup \{B \in A_o(B_i)/P_1(u(B)) < P_1(u(B_k))\}$.

I_5: optimality at upper corners. We show that after processing $u(B_i)$, $W[B_i]$ stores $W(B_i)$ (a Maximum Weighted Chain with a predecessor in $A_b(B_i) \cup A_o(B_i)$). As all predecessors of B_i are closed, let us denote by B, the right most predecessor of B_i on the x-axis: $B := \arg\max_{B_j \ll_r B_i}(P_1(u(B_j)))$.

1. If $u(B) \in A_b(B_i)$ then all predecessors of B_i are contained in $A_b(B_i)$. Hence, this situation was handled when processing $l(B_i)$, and Invariant I_4 regarding the *partial optimality at lower corners*, ensures that $W[B_i]$ stores $W(B_i)$.

2. If $u(B) \in A_o(B_i)$, $W[B_i]$ has been correctly updated (lines 14-18), while B_i was open, when sweeping $u(B_k)$ for each box $B_k \in \mathcal{B}$ such that $B_k \ll_r B_i$ and $u(B_k) \in A_o(B_i)$.

Hence, all predecessors of B_i have been taken into account, and $W[B_i]$ stores $W(B_i)$. This concludes the proof of I_5, and closes the correctness proof.

5.4.2 Time and space analysis

The running time and space analysis of Algorithm 6 is similar to that of Algorithm 2 in Section 4.6 for the two-dimensional case. However, in the case of our algorithm we

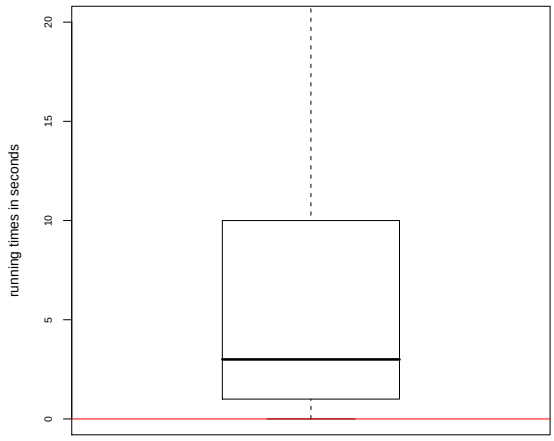


Figure 5.6: Running times in seconds of our algorithm (Algorithm 6) presented with a box plot (see Section 3.6.2): in 75% of the cases the algorithm needs less than 10 seconds. The box plot was truncated because of the 30 outlier cases that made the plot completely unreadable. In these 30 cases the algorithm needed between 3 and 54 minutes, as it had to deal with > 1 million fragments.

were not able to prove a $O(n \log n)$ time complexity, therefore we upper bound it to $O(n^2)$.

Obviously, the sets $\mathcal{O}$ and $\mathcal{A}$ contain at most n boxes, and thus require together with arrays Pred[.] and W[.], $O(n)$ space. We use balanced binary search trees (BST) to store $\mathcal{A}$ and $\mathcal{O}$, with boxes at the leaves ordered on $P_2(u(.))$, resp. $P_1(l(.))$. Hence, the amortised time needed for all insertions, deletions, and rebalancing is $O(n \log n)$ (Knuth, 1998, chap. 6). However, looking for the active boxes that can be deleted at each execution of the outer loop (lines 19-21) may force us to examine all boxes in $\mathcal{A}$. As this is the more complex operation in the outer loop, we obtain in the worst case an $O(n^2)$ time and $O(n)$ space complexity. Algorithm 6 maintains the subset of potential predecessors in $\mathcal{A}$ instead of searching through the whole box set as in Algorithm DP, which makes the practical difference.

The experimental running times observed when performing 694 whole genome pairwise comparisons (described in Chapter 6) show that this optimisation yields significant improvements: Algorithm 6 needed ≈ 8 hours to compute the 694 chains, while Algorithm DP took 3.5 days for the same computation. The improvement increases with the number of input fragments, and becomes considerable above $50,000$ fragments. In fact, below $50,000$ fragments, both Algorithm 6 and Algorithm DP take seconds, *i.e.* less than 1 minute. However, for n ranging from $50,000$ to $100,000$ Algorithm 6 still takes less than a minute, while Algorithm DP needs between 1 and

6 minutes to compute the chain.

Moreover, the gap between the two algorithms is widening beyond the threshold of $100,000$ fragments. The following examples illustrate this statement: for a pair of strains (CP000011 vs CP000548) from *Burkholderia mallei* species for which we computed $144,685$ fragments, Algorithm DP takes ≈ 16 minutes, while Algorithm 6 only needs less than 1 minute; for two strains from *Burkholderia pseudomallei* species (CP000573 vs CP000125) with $n = 197,310$ fragments, Algorithm DP takes 34 minutes and Algorithm 6 less than 3 minutes; for *Neisseria meningitidis* species (AM421808 vs AE002098) having $n = 307,852$, Algorithm DP needs 57 minutes and Algorithm 6 only 7 minutes. Finally, for a couple of strains (BX248333 vs AM408590) from *Mycobacterium bovis* with $1,000,000$ fragments, Algorithm 6 needs ≈ 1 hour, while Algorithm DP ended after 12 hours. More details on the running times of Algorithm 6 can be found in Figure 5.6.

5.5 Conclusion

As we have seen in the last two chapters, several versions of the chaining problem exist, *i.e.* generalizations of the original version of the chaining problem. This original problem, known under the name of *collinear chaining*, consists of selecting a best weighted subset of fragments that are collinear and do not overlap on any of the compared sequences, *i.e.* a chain. All the other versions of the chaining problem usually relax the collinearity constraint and, by abuse of language, we may still refer to the subset of fragments they compute, as chain.

For the collinear chaining problem we have seen that a quadratic time dynamic programming algorithm can be easily applied by using a recurrence with respect to the partial order between fragments. In order to speed up this algorithm, multiple formulations and solutions for this problem have been proposed in the literature, usually based on the technique of *sparse dynamic programming*. With this optimization, they manage to obtain an $O(n \log n)$ time complexity for the two-dimensional case.

Because of large scale alignment of genomic sequences, collinear chaining has become insufficient and taking rearrangements into account is now mandatory. Unfortunately, when taking rearrangements into account, the problem becomes NP-complete. Therefore one needs to use a special type of chaining with rearrangements, *i.e.* glocal chaining, or heuristics adapted to this particular case. If the collinear chaining problem is practically closed nowadays, for chaining with rearrangements there is a lot of place for improvements. Even though glocal chaining seems a promising lead, only one study on this problem exists and, for the moment, it has no mathematical basis, no multi-dimensional extension and several interrogation points regarding the biological relevance remain.

Moreover, we have seen that in order to maximize the length of the chain, one would like to allow some of the overlaps between adjacent fragments. There is little work on this subject and the only methods taking overlaps into account, allow them without restriction or trim fragments in order to remove them. The only method that we found

that explicitly deals with overlaps, (Shibuya and Kurochkin, 2003), fixes a maximal overlap threshold, which proves to be insufficient because of variable overlap lengths. In conclusion, in order to fulfil new needs in computational biology, we extended the classical framework of Maximum Weighted Chain by authorizing overlaps between fragments in the computed chain, and formalized the MWC with Proportional Length Overlap problem where overlaps are proportional to the fragment lengths. Difficulties arise from the fact that the weights of overlaps are deduced from the chain weight.

In (Uricaru et al., 2010), we exhibited the first two algorithms for this problem, which both solve it in quadratic time in function of the number of fragments. It is obvious that due to our definition of a collinear chain, we obtain chains of fragments that are longer than those not allowing overlaps. The question of whether our results are truly significant from a biological point of view is to be discussed in Chapter 6. At that point we examine the experiments on real data sets, and showed that i/ overlaps significantly improve global genome results, ii/ our sweep line algorithm outperforms the truly quadratic dynamic programming solution in practice. Thus, we may say that allowing for overlaps improves the global genome alignment results significantly, at a reasonable cost. However, the study of the average time complexity of the sweep line algorithm remains open. Comparing with fixed overlaps, as well as investigating the robustness of chains with respect to the ratio of allowed overlaps are future lines of research.

6 Personal contribution. Applying chaining with proportional overlaps to WGA

We propose a novel WGA method that combines the advantages of local similarities (LS) with the ones of the chaining method with proportional overlaps, described in the previous chapter. This new pairwise whole genome alignment tool named YOC is based on a less complex strategy than classical WGA programs. It implements a **two phase strategy**: *fragment computation* combined with *adapted chaining*, thus renouncing to the classical recursivity and "last chance alignment" phases.

Contents

6.1 Further study on the impact of using local similarities as fragments

In this chapter, we continue the experiments from Chapter 3. However, note that the results presented in the current chapter were obtained on a different dataset than the one described in Section 3.2, as already explained at that time.

Dataset 2 This dataset extends and slightly modifies the first one, *i.e.* *dataset* 1. It includes 87 different species (346 different genomes) representing 694 intra-species pairs. Once again, we consider all pairs of bacterial strains of the same species extracted from the Genome Reviews database, this time at mid-2009, like in the release 5.0 of the Mosaic database.

- 588 pairs (85%) were inserted in the MOSAIC database according to the criterion of 50% of minimum coverage of the backbone regions compared to the average length of genomes;
- 174 pairs (25%) were considered as being collinear according to the criteria described in (Chiapello et al., 2008), *i.e.* no inversion or translocation were detected with Mauve aligner, or none of the inverted or translocated segments detected by Mauve exceeded a threshold of 20 kb in length.

By pairing local similarities with an adapted type of chaining in a new strategy for WGA, we are able to avoid the problems encountered in Section 3.6.3 and study LS' true advantages compared to exact matches. We begin by analyzing the behaviour of our novel chaining method with proportional overlaps, *i.e.* OverlapChainer, compared to the classical collinear chaining, *i.e.* Chainer. Second, having an exact method for chaining with overlaps allows us to do accurate tests on the true influence of spaced-seed based local similarities compared to exact matches and to local similarities based on contiguous seeds. This new two-phase strategy for WGA gave a WGA tool that we name YOC. In Section 6.3, we conduct several global performance comparisons to other four WGA tools, based on a protocol described in Section 6.2. We also describe a protocol for biological evaluation based on orthologous genes in Section 6.4. The biological evaluation is ongoing but some initial results are presented in Section 6.5.

6.1.1 Impact of the chaining algorithm

We first compare the results between six different methods, three using the collinear chaining algorithm Chainer: VC, BC, YC (see details in Section 3.6) and three relying on the OverlapChainer method: VOC (VMatch-OverlapChainer), BOC (Blast-OverlapChainer), YOC (YASS-OverlapChainer), see Figure 6.1.

We computed the differences of coverage and identity percentages between the six methods on *dataset* 2 composed of 694 pairs of genomes described above.

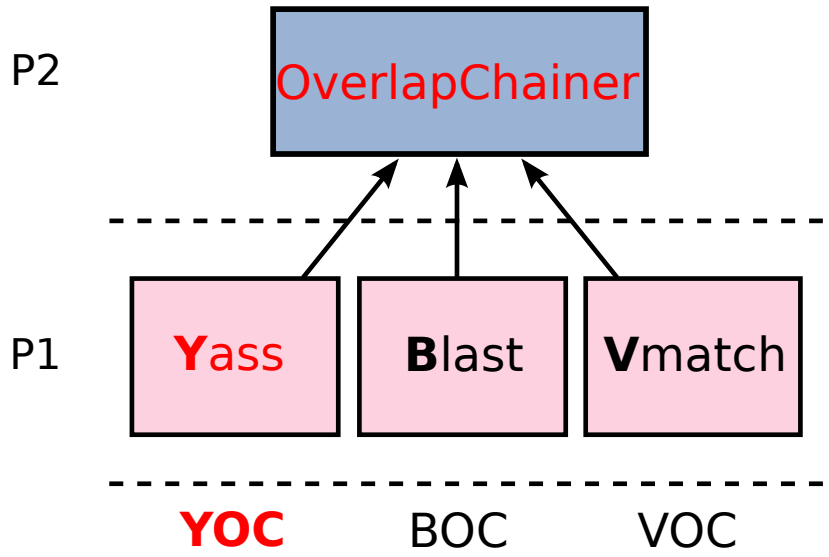


Figure 6.1: Three novel prototypes composed of two phases, *i.e. fragments computation + chaining with proportional overlaps*, namely: VOC for VMatch + OverlapChainer, BOC for Blast + OverlapChainer and YOC for YASS + OverlapChainer.

According to its definition, we expect the chaining allowing for overlaps combined with the three computation fragments methods to give coverages and identity percentages superior or equal to those obtained with the classical chaining. Moreover, results of chaining allowing overlaps should be dramatically improved in the case of local similarities, compared to other types of computing fragments methods, as they are the ones more likely to overlap. In order to confirm these assumptions, we compared the following three pairs of tools: VC to VOC (VOC−VC), BC to BOC (BOC−BC) and YC to YOC (YOC−YC), see Table 6.1 and Figure 6.2 for the distribution of coverage and identity percentages differences.

As expected, for local similarities, results were largely improved by using the chaining algorithm with proportional overlaps, *i.e.* OverlapChainer, instead of the classical chaining (average at 16.8 and 17, for identity percentage, respectively cov% differences). For example, when comparing strains $CP000046$ and $BA000018$ of *S. aureus* discussed in Section 3.6.3, YC obtains a cov% of 65%. YOC, however, reaches a coverage percentage of 94%. Remember that in this case 14 holes in the classical chain have not been filled due to fragments overlapping, see page 83 in Section 3.6.3.

In the case of BOC−BC and VOC−VC, as Blast and VMatch produce shorter fragments thus diminishing the impact of overlaps, the improvement of OverlapChainer is less important. However, we can still account for 3.4%, respectively 2.4% of id% improvement in average (3.4%, respectively 1.6% of cov% improvement in average).

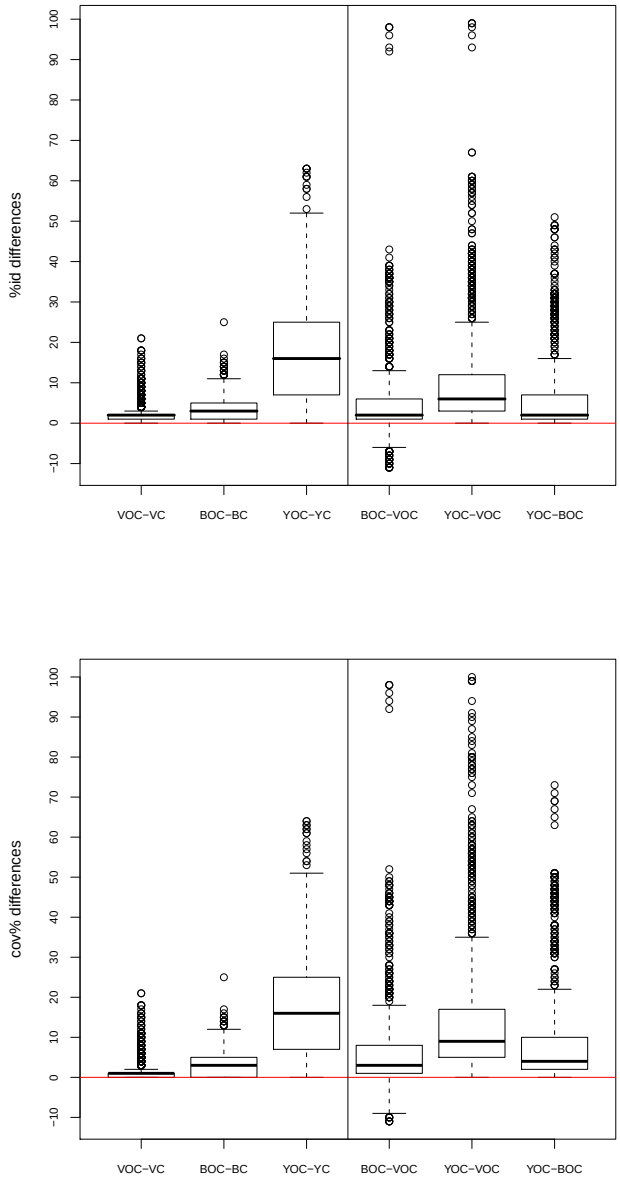


Figure 6.2: Differences of id% (upper image) and coverage percentages (lower image), presented as box plots, between six methods on a dataset of 694 pairs of aligned bacterial genomes. Compared methods are VC (VMatch-Chainer), BC (Blast-Chainer), YC (YASS-Chainer) and VOC (VMatch-OverlapChainer), BOC (Blast-OverlapChainer), YOC (YASS-OverlapChainer).

id% diff	minimum	maximum	average
VOC−VC	0	21	2.4
BOC−BC	0	25	3.4
YOC−YC	0	63	16.8
cov% diff	minimum	maximum	average
VOC−VC	0	21	1.6
BOC−BC	0	25	3.4
YOC−YC	0	64	17

Table 6.1: Differences of identity and coverage percentages between six methods on a dataset of 694 pairs of bacterial genomes (minimum, maximum, average). Compared methods are VOC vs VC, BOC vs BC and YOC vs YC.

id% diff	minimum	maximum	average
BOC−VOC	−11	98	4.8
YOC−VOC	0	99	11.2
YOC−BOC	0	51	6.4
cov% diff	minimum	maximum	average
BOC−VOC	−11	98	6.2
YOC−VOC	0	100	15
YOC−BOC	0	73	8.9

Table 6.2: Differences of identity and coverage percentages between six methods on a dataset of 694 pairs of bacterial genomes (minimum, maximum, average). Compared methods are BOC vs VOC, YOC vs VOC and YOC vs BOC.

6.1.2 Impact of the seed type

In order to show the impact of spaced seed based local similarities, independently of the chaining method, we compared three tools (among the six) that use the chaining with proportional overlaps: VOC to BOC (BOC−VOC), VOC to YOC (YOC−VOC) and BOC to YOC (YOC−BOC), see Table 6.2 and Figure 6.2. Our method based on local similarities computed from spaced seeds, YOC, obtains in average better results than the other two, *i.e.* 11.2% and 6.4% additional identity percentage compared to VOC, respectively BOC. Moreover, YOC can even improve on VOC and BOC with more than 50% (both identity and coverage percentages) in several cases.

Therefore, local similarities (computed with spaced seeds, YASS, or with inexact matches, Blast), together with an adapted chaining, stand out as a better alternative to short exact or inexact matches. Moreover, local similarities computed with spaced seeds prove to be an even better alternative than those computed with inexact matches

(6.4% additional id% in average). Finally, the chaining with proportional overlaps is a novel chaining method perfectly adapted to local similarities.

Further conclusions on the use of local similarities with an exact algorithm for chaining with overlaps One may argue that proving local similarities to be better than short exact or inexact matches, is a predictable result, as it is obvious that local similarities cover the genomes in a greater proportion than short matches. However, this initial study allowed us to identify the specific problems brought by this kind of fragments and to precisely assess the induced amount of improvement.

Next, we wanted to see to which extent the last two phases contribute to the coverage and the identity percentage of the final alignment. Thus we consider YOC, which implements only two phases, as an alternative to standard WGA tools. This is based on the idea that long local similarities computed in one step could replace exact or approximate matches computed in several recursive steps. Results in the following sections, based on the comparison of our best prototype YOC, now a standalone method, to four-phase WGA tools shed light on this issue.

6.2 Protocol for YOC global performance evaluation, compared to four WGA tools

We compare our method, YOC, to four genome alignment tools, on *dataset* 2. Even though we have seen in Section 2.6.3 that comparing the results of WGA tools is a standalone problem far from being solved, we propose a global evaluation protocol that should give us an idea on the performances of each method.

Methods The YOC algorithm relies on two simple ideas: (1) enhancing the computing fragments step, by *using local alignments* instead of short matches as anchors (2) usage of *a new chaining algorithm that takes into account overlapping fragments*. Observe that YOC method implements only the first two phases of the anchor strategy, see Section 6.1.1, avoiding in this manner the recursion and the "last chance alignment phase" and thus the errors due to these phases.

YOC was clearly designed for dealing with collinear genomes, as it is based on a collinear chaining phase. However, YOC may be considered an intermediate algorithm, as it allows to detect and align "locally" inversed regions (but not translocated segments): fragments located on the reverse strand are reversed in order to consider them in the collinear chain, see Section 3.6.2.

YOC is compared to four WGA tools: three multiple alignment tools described in Section 2.5, *i.e.* MGA (Hohl et al., 2002), Mauve (Darling et al., 2004) and ProgressiveMauve (Darling et al., 2010), and one pairwise alignment tool described in Section 2.4, LAGAN (Brudno et al., 2003b). Even though three of these tools were designed for multiple alignment, in our experiments they are only employed for pairwise comparisons. In fact, their strategy suits in an equal measure the pairwise and

the multiple alignment problems.

First, note that MGA, Mauve, ProgressiveMauve and LAGAN are all variations on the four phase anchor based strategy. However, ProgressiveMauve executes an additional phase meant to filter low quality alignment regions.

Moreover, two of these four tools are clearly addressed to collinear genome comparisons (MGA and LAGAN), while two others are able to align rearranged genomes, *i.e.* including inversions or translocations (Mauve and ProgressiveMauve). Regarding rearrangements, YOC may be placed in between these two categories.

Other tools, like M-GCAT (Treangen and Messeguer, 2006) and Shuffle LAGAN (Brudno et al., 2003c) selected at first for our experiments, were eliminated from our set of tested tools as M-GCAT is an extremely close relative of Mauve, which in addition obtains poor results on divergent couples, and Shuffle LAGAN is not always able to cope with complete genomic sequences due to the high number of fragments in the chaining phase.

Parameters and comparison strategy MGA and LAGAN, tools designed for collinear comparisons, were used to process the 174 collinear pairs. Mauve, ProgressiveMauve and YOC processed the entire dataset of 694 couples of bacterial genomes.

MGA and Mauve raw alignments were extracted from the release 5.0 of Mosaic db; they were obtained with the parameters described in Section 3.2.1. YASS parameters were described in Section 3.6.2 and OverlapChainer was applied with 0.1 ratio. We used LAGAN version 1.1 and ProgressiveMauve version 2.3.1. For LAGAN we employed the default parameter settings as detailed in (Brudno et al., 2003b). For ProgressiveMauve the parameter settings were: *max-gapped-aligner-length* = 10000 and *weight* = 5000 as for Mauve, and the rest were left on the default values as they were calibrated for divergent bacterial sequences. In fact, compared to Mauve, ProgressiveMauve needs much less adjustment of parameters.

Based on the results described in Section 3.5 that questioned the quality of some regions in the alignment, we decided to filter low quality regions. For this, we chose GRAPe software (Lunter et al., 2008), see Section 2.6.2. The same *filtering procedure*, described in Section 3.4, was used on raw pairwise alignments of all five WGA tools. Alignments after filtering are what we call from now on *filtered backbones*. Observe that due to its final filtering phase, filtering ProgressiveMauve alignments is redundant but we applied it anyway, as to use the same exact protocol for all tools.

We analyzed the global genome alignment results before and after the filtering (*i.e.* backbones derived from raw alignments, by studying the differences of identity and coverage percentages as in the previous sections.

Filtered cov% & id% for the 174 collinear cases						
tools	minimum		maximum		average	
	cov%	id%	cov%	id%	cov%	id%
YOC	0	0	4	1	0.81	0.18
MGA	0	0	13	7	2.55	1
LAGAN	0	0	21	3	5	1.08
Mauve	0	0	35	23	2.44	0.57
ProgressiveMauve	0	0	1	1	0	0
Filtered cov% & id% for the 520 non-collinear cases						
YOC	0	0	3	1	0.74	0.22
Mauve	0	0	56	34	5.21	1.62
ProgressiveMauve	0	0	1	1	0.01	0.01

Table 6.3: The amount of coverage and identity percentages (minimum, maximum, average) filtered by GRAPe for YOC, MGA, LAGAN, Mauve and ProgressiveMauve.

6.3 Results of YOC global performance evaluation

6.3.1 Study on the amount of coverage and identity percentages filtered

First we analyzed the amount of identity and coverage percentages filtered by GRAPe for each of the five tools, on both the collinear and the non-collinear datasets. As one may observe in Table 6.3 and in Figures 6.3 and 6.4, GRAPe filtering affects the coverage significantly more than it affects the identity percentages, for both collinear and non-collinear cases, meaning that GRAPe truly removes low quality alignment regions.

YOC and ProgressiveMauve have in average less than 1% of the entire genomes filtered from the alignments (generating little changes in coverages and identity percentages) thus indicating a high quality of their alignments. This can be explained by the statistical significance of the local similarities generated by YASS and the filtering phase of ProgressiveMauve (avoiding errors due to the last phases of the anchor-based strategy).

MGA, LAGAN and Mauve have in average more than 2% of the genomes filtered (and in some cases, more than 10%), indicating that even if MGA, LAGAN and especially Mauve can reach extremely high coverages, they are in part due to low quality regions, mostly added in the fourth phase of the anchor-based strategy. Biological results in Section 6.5 corroborate these observations.

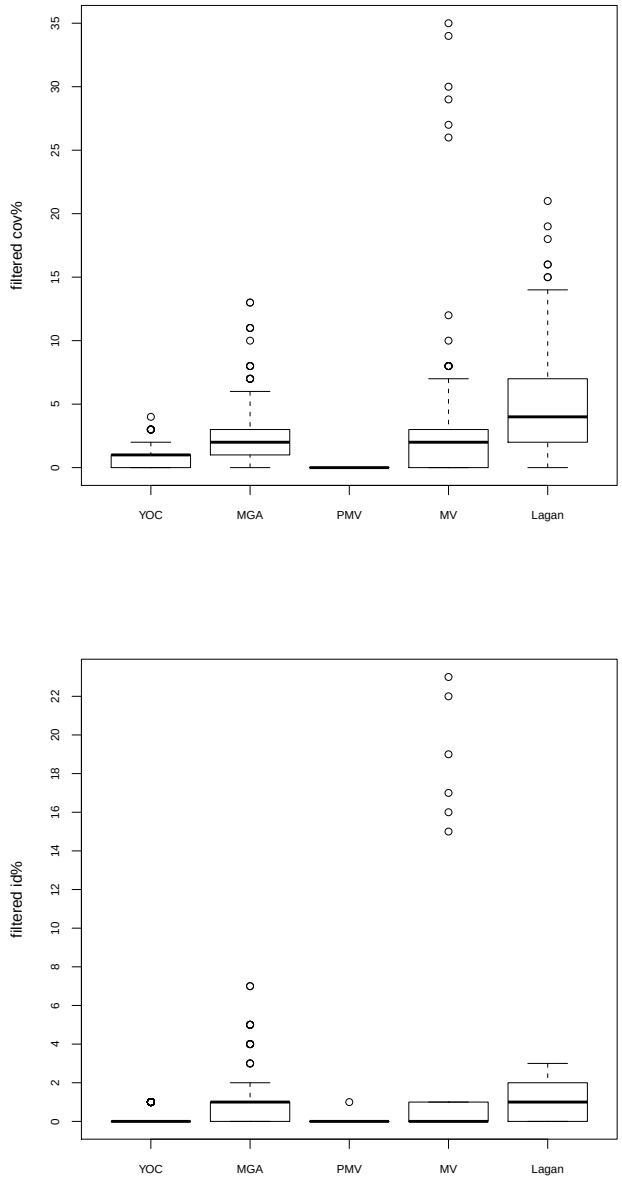


Figure 6.3: **Collinear pairs** Box plots summarizing the amount of coverage (upper image) and identity percentages (lower image) filtered by GRAPe for YOC, MGA, ProgressiveMauve, Mauve and LAGAN, on a dataset of 174 collinear pairs of bacterial genomes. Observe that YOC and ProgressiveMauve have very little amount of cov% and id% filtered, while MGA, LAGAN and especially Mauve may reach extremely high filtering percentages.

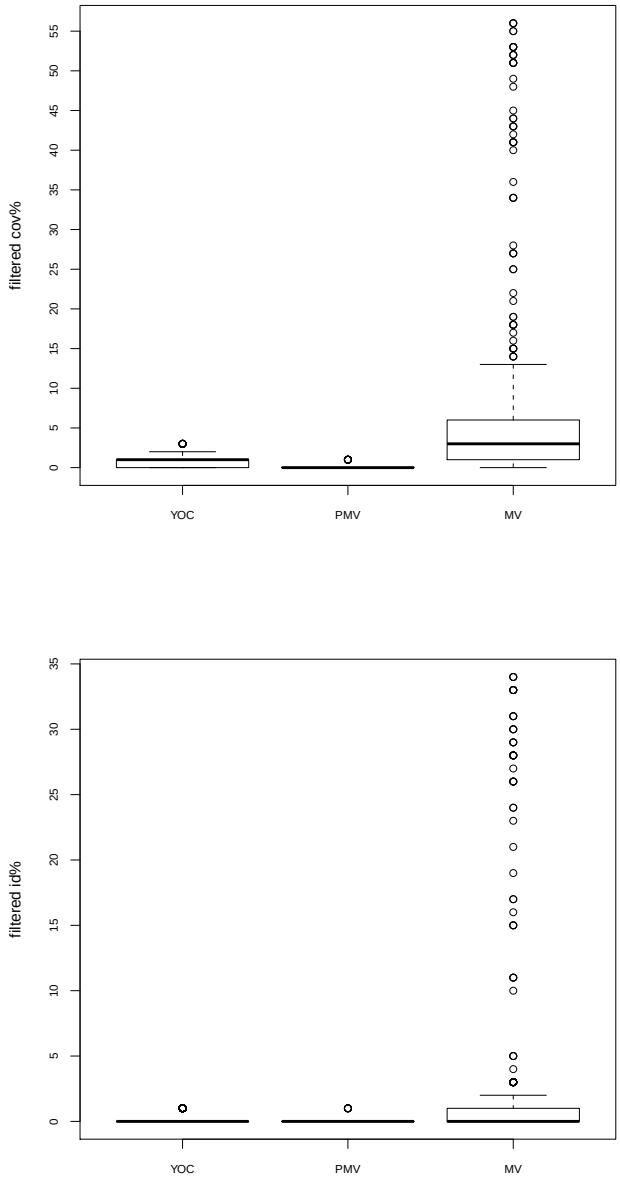


Figure 6.4: **Non-collinear pairs** Box plots summarizing the amount of coverage (upper image) and identity percentages (lower image) filtered by GRAPe for YOC, ProgressiveMauve and Mauve, on a dataset of 520 rearranged pairs of bacterial genomes. Observe that YOC and ProgressiveMauve have very little amount of cov% and id% filtered, while Mauve may reach extremely high filtering percentages.

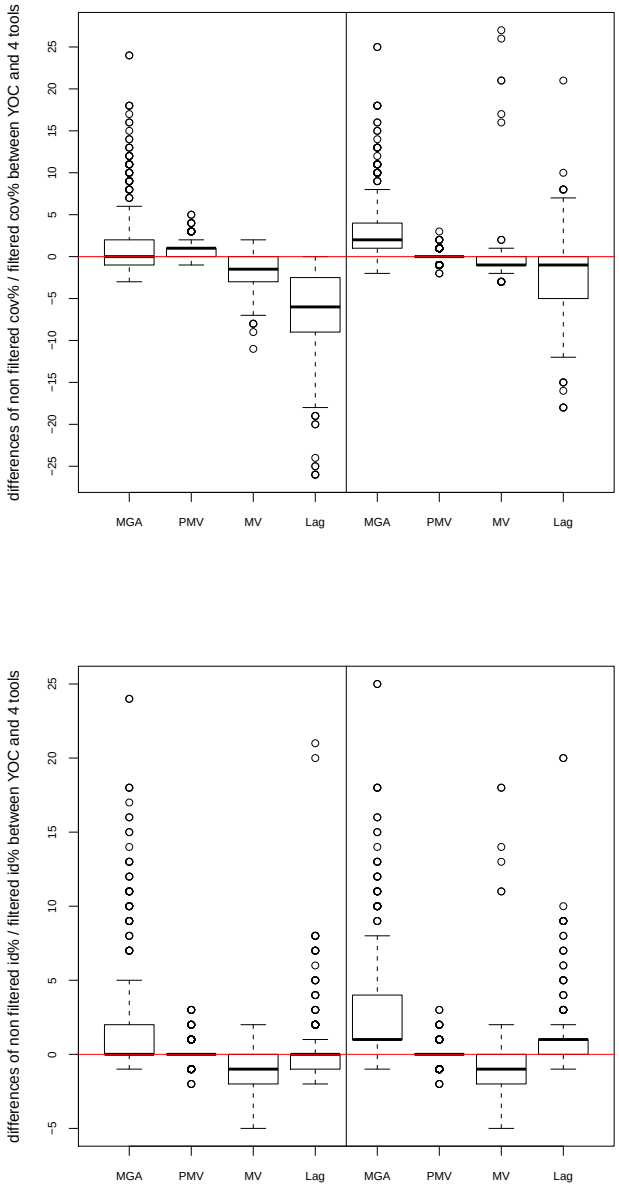


Figure 6.5: **Collinear pairs** Differences of coverage (upper part of the image) and identity percentages (lower part of the image), before and after GRAPe filtering, between YOC and another four methods on a dataset of 174 collinear pairs of bacterial genomes. The methods that we compare to YOC are: MGA, LAGAN, Mauve and ProgressiveMauve. Observe the important difference between cov% and id% results, both before and after the filtering.

id% diff before filtering	minimum	maximum	average
YOC−MGA	−1	24	2.09
YOC−LAGAN	−2	21	0.26
YOC−Mauve	−5	2	−1.3
YOC−ProgressiveMauve	−2	3	0.11
id% diff after filtering			
YOC−MGA	−1	25	2.85
YOC−LAGAN	−1	20	1.16
YOC−Mauve	−5	18	−0.91
YOC−ProgressiveMauve	−2	3	−0.06

Table 6.4: **Collinear pairs** Differences of identity percentages, before and after GRAPe filtering, between YOC and another four methods on a dataset of 174 collinear pairs of bacterial genomes (minimum, maximum, average). The methods that we compare to YOC are: MGA, LAGAN, Mauve and ProgressiveMauve.

6.3.2 YOC compared to MGA, LAGAN, Mauve and ProgressiveMauve

In Table 6.4 and Table 6.5 we compared YOC to the other methods, on the collinear dataset, respectively on the non-collinear one. As YOC is mostly a method for collinear genomes (not dealing with translocations, nor with complex inversions), results distribution changes a lot between collinear and non-collinear pairs.

On the **collinear dataset**, see Table 6.4 and Figure 6.5, YOC obtains very good results. It has a slight advantage for the average identity percentage when compared to MGA and LAGAN, both before and after the filtering, advantage that can reach more than 20% for pairs of divergent genomes. On the same dataset, YOC obtains results that are comparable to those of ProgressiveMauve, and slightly worse than those of Mauve in average.

As one may notice in Figure 6.5, regarding the cov%, YOC results are very low compared to those of LAGAN. This can be explained based on the particularity of LAGAN, which forces a complete alignment of the regions, *i.e.* leaves no unaligned regions. In fact, when examining the id% results, we observe a completely different configuration: YOC has better results than LAGAN, especially after filtering. This suggests that LAGAN wrongly aligns sequence regions in order to maximize a collinear complete global alignment.

On the **non-collinear dataset**, see Table 6.5 and Figure 6.6, YOC is obviously in difficulty when compared to Mauve and ProgressiveMauve, two tools especially designed for non-collinear alignment. However, several observations can still be made.

As for the collinear cases, for non-collinear cases GRAPe filters very little from YOC

id% diff before filtering	minimum	maximum	average
YOC−Mauve	−71	14	−13.75
YOC−ProgressiveMauve	−57	49	−9.54
YOC−ProgressiveMauve on *B. aphidicola*	8	19	11.75
id% diff after filtering			
YOC−Mauve	−71	21	−12.35
YOC−ProgressiveMauve	−57	49	−9.75
YOC−ProgressiveMauve on *B. aphidicola*	8	19	11.75

Table 6.5: **Non-collinear pairs** Differences of identity percentages, before and after GRAPe filtering, between YOC and another two methods on a dataset of 520 non-collinear pairs of bacterial genomes (minimum, maximum, average). The methods that we compare to YOC are: Mauve and ProgressiveMauve. A snapshot of the *B. aphidicola* case.

alignments, therefore alignments are mostly correct but they are probably not complete, because of the rearrangements that could not be accommodated. One cannot say the same thing for Mauve that has 5.21% of cov% filtered in average, as seen in Table 6.3.

Moreover, for extremely divergent cases, YOC obtains better results than the other two methods, *e.g. B. aphidicola*. Indeed, for the six pairs of *B. aphidicola*, YOC has in average 11.75% more than the identity percentage of ProgressiveMauve (with a maximum at 19%), both before and after the filtering, see Table 6.5.

6.3.3 Conclusion for the global evaluation of YOC method

Based on the evaluation described in Section 6.1 and on the global evaluation in the current section, we can make several observations on our new WGA method, YOC:

- the anchor chains computed by anchor based methods can be improved by using local alignments instead of short matches, provided that the chaining algorithm authorizes overlaps between adjacent anchors. This improvement measured in terms of genome coverage and of id% is even more pronounced if local alignments are detected with highly sensitive spaced seeds. As most anchor-based programs use the same type of similarities as MGA, Mauve, LAGAN and ProgressiveMauve, our conclusion is likely to be general and should help improving multiple alignments as well.

- as already suspected, unreliable parts of MGA, Mauve, ProgressiveMauve and LAGAN alignments are generated by the fourth, “last chance alignment” phase, which tries to align pairs of regions in which no anchors were found by the previous phases. In fact, if Mauve or LAGAN often achieve higher genome

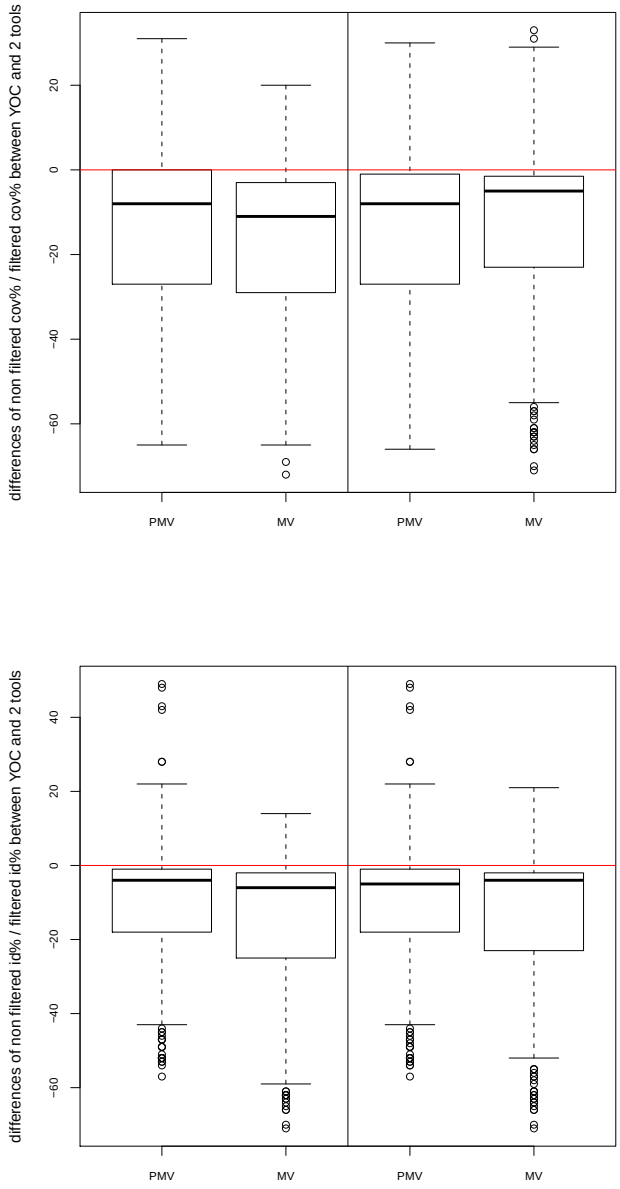


Figure 6.6: **Non-collinear pairs** Differences of coverage (upper part of the image) and identity percentages (lower part of the image), before and after GRAPe filtering, between YOC and another two methods on a dataset of 520 non-collinear pairs of bacterial genomes. The methods that we compare to YOC are: Mauve and ProgressiveMauve.

coverages than YOC, the alignments they output usually require a "cleansing" before attempting a biological interpretation. ProgressiveMauve on the other hand has a filtering phase that removes these low quality regions, thus its results

are to be trusted, see Section 2.5.2.

YOC obtains good alignment solutions, similar to ProgressiveMauve, but this is done by using a different strategy: less complex, *i.e.* with less phases and less parameters to configure. In fact, it directly computes trustworthy results, without applying an additional filtering phase. For both ProgressiveMauve and YOC, such a filtering phase has little influence on their results.

- it is true that YOC has an handicap in front of Mauve and ProgressiveMauve because it does not deal with rearrangements. This is why, results on non-collinear pairs are not so encouraging. However, for collinear pairs, YOC obtains similar results to those obtained by the most accurate method among the four methods, ProgressiveMauve, and better results than the other three. Interestingly, YOC performs drastic improvements where the other tools fail, including ProgressiveMauve: on species with highly divergent strains like *B. aphidicola* or *P. marinus*.

- in addition, YOC chain contains 150 anchors in average vs several thousands for MGA, Mauve or ProgressiveMauve (LAGAN gives a complete global alignment thus it cannot be analyzed in this manner) making it simpler to visualize and to grasp.

6.4 Protocol for biological evaluation based on orthologous genes

In this section, we explain the protocol established for estimating the reliability and robustness of WGA of bacterial genomes from a biological view-point.

One ideally expects orthologs to be properly aligned in backbone segments (BS). To assess the biological relevance of alignments, we verified whether orthologous genes were completely included in the backbone, and if their position is consistent on the two genomes. The pairs of orthologs and their genomic positions were taken from the OMA database (Schneider et al., 2007). The set of genome pairs having the same accession numbers in our *dataset* 2 and in OMA contains 161 pairs from 34 species. The results are summarized in Section 6.5.

6.4.1 OMA database

OMA db is one of the largest orthology projects, continuously updated, which unlike similar projects like COGs database (Tatusov et al., 2000) or KEGG (Ogata et al., 1999), the database used in (Swidan and Shamir, 2009) for a purpose similar to ours, does not rely on human intervention. In fact, once a genome is integrated into OMA db, the process is fully automated. The OMA project is based on a massive cross-comparison of complete genomes coming from all kingdoms of life, meant to identify the evolutionary relation between any pair of proteins. One of the most

notable qualities of the project consists in its *strictness*: in undecided cases (lacking of discriminating information), it favours missing orthology relation classification, over erroneous orthology assignment. More details can be found on OMA Browser web page and in (Schneider et al., 2007).

6.4.2 Position of orthologous genes on the filtered backbones

The assumption underlying our evaluation is that orthologous genes, which are by definition shared by the strains in comparison, should be located in conserved regions of their genomes. Thus, we expect orthologous genes to be aligned generally over their full length in what we call the backbone segments of these genomes.

We took the filtered backbones obtained on the 161 pairs by three tools: YOC, MGA and ProgressiveMauve, as they are the ones with the most interesting results, each of them being adapted to a type of genome comparison. Mauve and LAGAN were left out of our testing set as, based on the results of the previous section, we considered that their alignments were not reliable enough. For each one of the filtered backbones among the 3×161 ones, we use the following two measures, meant to evaluate the quality of the segmentation of the filtered backbones on every pair of genomes.

First measure On each genome from an aligned pair of genomes, we compute the number of complete orthologous genes as well as the number of nucleotides from orthologous genes that were aligned, which are:

- included in a segment of the filtered backbone;
- included in a region that is outside the filtered backbone (corresponding to variable segments);
- overlapping two segments, one in the filtered backbone, and a second one outside of it.

See Figure 6.7 for a graphical representation of these three situations.

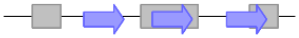


Figure 6.7: Genes are represented by arrows and variable segments by grey blocks. Segments between blocks correspond to segments of the filtered backbone. The first arrow corresponds to an ortholog completely included in a segment of the filtered backbone, the second arrow is an ortholog placed in a variable segment and the third one is overlapping two segments.

This first measure verifies whether the segmentation produced on one genome is precise, *i.e.* if a gene has even one position in a different segment, the entire gene is counted as not being completely included in the filtered backbone. Moreover, it analyzes whether the segmentation is globally correct, *i.e.* by counting the number of nucleotides composing the genes, included in backbone segments.

Second measure For every pair of orthologs placed in a segment of the filtered backbone on at least one of the two genomes, we test whether their positions on the two genomes are consistent, meaning whether the orthologs are:

- included in the same segment of the filtered backbone on both genomes, *i.e.* *identical positions*;
- included in different segments of the filtered backbone on the two genomes, *i.e.* *different positions*;
- included in a segment of the filtered backbone on one of the genomes and overlapping a backbone segment and a variable segment on the second one, *i.e.* *overlapping positions*.

See Figure 6.8 for a graphical representation of these three situations.

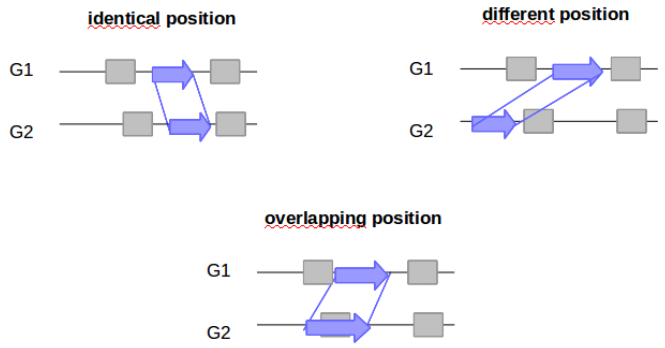


Figure 6.8: The two parallel lines represent the two genomic sequences with their alignment. Genes are represented by arrows and variable segments by grey blocks. Segments between blocks correspond to segments of the filtered backbone. The first image corresponds to a pair of orthologs with identical positions, *i.e.* placed in the same backbone segment on the two genomes. The second image corresponds to a pair of orthologs with different positions in the two genomes. The last image represents a pair of orthologs with overlapping positions on one of the genomes.

This second measure verifies the consistency of the backbone on the two genomes.

In Section 6.5, we give a summary of the results obtained for these measures and present several case studies.

6.4.3 Classification of genome pairs based on the coverage of the filtered backbone

Moreover, we wanted to analyze the quality of the segmentation in relation to the coverage of the filtered backbone. In order to do this, we classified the 161 pairs of genomes based on their coverage, as follows:

- pairs on which the results of the tools were almost identical ($\leq$ 2 cov% difference on both genomes);
- pairs on which ProgressiveMauve obtains the best results ($>$ 2 cov% difference on both genomes);
- pairs that cannot be associated to any category (*e.g.* YOC obtains better coverage than all the others on one genome and on the second genome it is ProgressiveMauve that gets the best coverage result)

Based on this classification of pairs we observe, as expected, that: ProgressiveMauve obtains the best results on non-collinear pairs corresponding to 96 cases among the 161, while on collinear pairs the results of the three tools are very similar, 62 pairs. For 3 pairs we could not choose a category.

Even though these results are in a preliminary phase, in Section 6.5 we attempt an analysis of the segmentation based on this classification.

6.5 Biological evaluation results

Regarding the analysis of the segmentation in relation to the classification of genome pairs based on the coverage (see Section 6.4.3), we can make the following observation.

For the first measure described in Section 6.4.2, when producing similar coverages, the three tools obtain similar results. This means that they completely include in the backbone the same number of orthologs, both in number of nucleotides and genes. When rearrangements involve a large part of the genome, ProgressiveMauve achieves better backbone coverages than MGA and YOC, and usually covers the same additional proportion of orthologous genes.

For the second measure on the other hand, we observe a different configuration. For 73 pairs among the 161, YOC obtains better results than the other tools. This means that YOC completely includes in the same backbone segments, on both genomes, more orthologs than the other tools, even when it reaches lower coverages, *i.e.* for divergent cases.

In fact, for divergent cases, YOC alignments can provide pronounced improvements compared to Mauve in terms of correctly aligned orthologous genes. If we take the pair of *P. marinus* strains (CP000552_GR vs CP000553_GR), ProgressiveMauve and YOC cover 62, respectively 53% of the genomes (9% of additional coverage for ProgressiveMauve), but YOC completely includes in backbone segments 46% of the 1366 orthologous genes, while ProgressiveMauve only 25%; moreover YOC completely includes in the same backbone segments, on both genomes, 68% of the 1366 orthologs, compared to only 29% for ProgressiveMauve.

Even though these results need to be further examined, we may assume that YOC backbone bounds are more accurate than those of MGA and ProgressiveMauve. This observation is comforted by the case study presented below.

6.5.1 Case study: *Lactococcus lactis* pair

In this section we discuss the results in terms of orthologous genes for the pair of *Lactococcus lactis* strains, IL1403 vs SK11, an example of divergent case.

In this case, both ProgressiveMauve and YOC cover $\simeq$ 76% of the genomes, but YOC completely includes in backbone segments 88% of the 1513 orthologous genes, while ProgressiveMauve only 85%. Indeed for ProgressiveMauve, 15% of orthologs are split into backbone and variable segments. Moreover YOC completely includes in the same backbone segments, on both genomes, 91% of the 1513 orthologs, compared to only 84% for ProgressiveMauve.

This phenomenon is clearly illustrated in Figures 6.9a & 6.9b, which show respectively the ProgressiveMauve & YOC alignments of the first, $34Kbp$, collinear block on both genomes, with their annotated genes. This block contains only four non-orthologous genes (yabC to yabF) in strain $IL1403$, and one in strain $SK11$. One observes that these non-orthologous genes are *hatched*, containing both alignable and unalignable regions (in green) in ProgressiveMauve alignment, while YOC detects two homologous, aligned regions separated by a single large insertion encompassing yabC to yabF genes (Figure 6.9b). Note that for ProgressiveMauve, some small aligned segments can be found inside the non-orthologous region; for the genome alignment problem, these small segments seem false positive subalignments. Moreover, a variable segment is found inside the homologous region.

To further illustrate the hatching of orthologs by ProgressiveMauve, we consider the yqjD gene. In Figure 6.10a, one sees that it is broken by ProgressiveMauve into 3 alignments. On the contrary, YOC yields a single $> 33Kbp$ alignment of 82% id%, which completely encompasses that gene, see Figure 6.10b. Note that this phenomenon, observed globally throughout the genome, is not due to the higher coverage of YOC, as YOC and ProgressiveMauve have similar coverages.

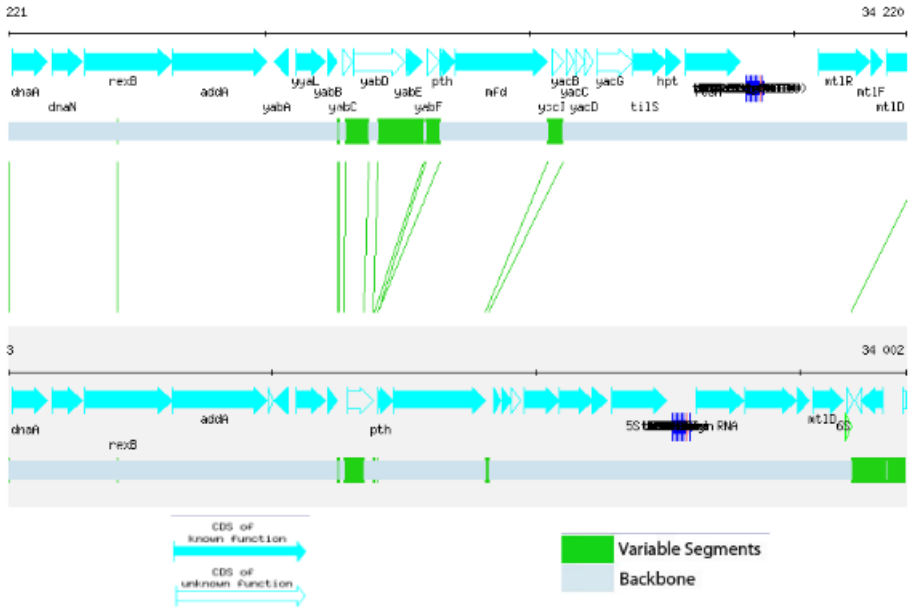
6.6 Conclusion

Based on initial experiments conducted in Chapter 3, we observe that local similarities have a specific problem: *frequent overlaps of various sizes*. We thus introduce a novel definition of collinear chaining, by allowing overlaps proportional with the lengths of the adjacent fragments, and propose an exact algorithm, called OverlapChainer, detailed in Chapter 5.

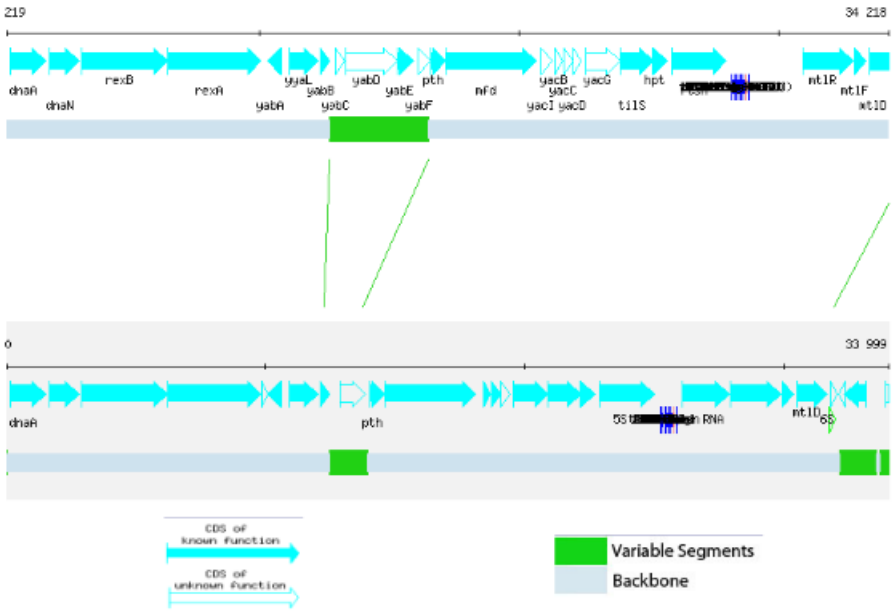
By using this novel chaining algorithm OverlapChainer, in Section 6.1, we show that *local similarities, together with an adapted chaining, stand out as a better alternative to short exact or inexact matches*. Moreover, local similarities computed with spaced seeds prove to be an even better alternative than those computed with inexact matches.

We thus obtained a two-phase WGA method, YOC, that we compare to several four phase anchor based tools (MGA, LAGAN, Mauve and ProgressiveMauve), based on global measures in Section 6.3, and biological measures in Section 6.5.

Unlike MGA, LAGAN, Mauve or ProgressiveMauve, YOC does not need fine tun-

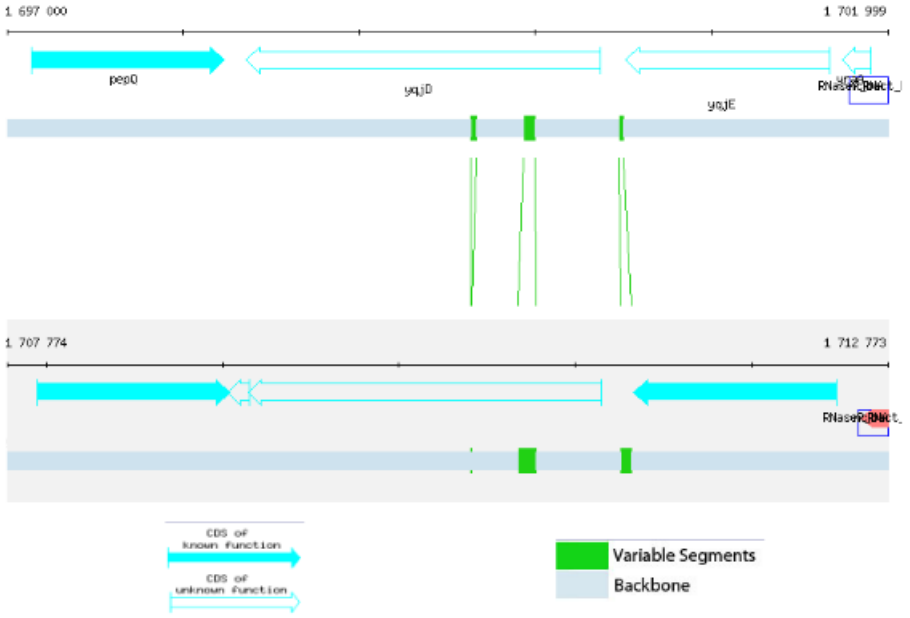


(a) ProgressiveMauve



(b) YOC

Figure 6.9: Backbone segmentation differences between ProgressiveMauve and YOC on a pair of *L. lactis* strains (IL1403 vs SK11). (a) & (b): ProgressiveMauve and YOC alignments of the first $34Kbp$ collinear block of the genomes. In each subfigure, the visualisation is done on the Mosaic server, strain IL1403 is at the top, SK11 at the bottom. Genes are displayed as arrows, and variable segments as green boxes.



(a) ProgressiveMauve: zoom on yqjD gene

(b) YOC: zoom on yqjD gene

Figure 6.10: A zoom on the yqjD gene region in (a) ProgressiveMauve alignment and (b) YOC alignment. In each subfigure, the visualisation is done on the Mosaic server, strain IL1403 is at the top, SK11 at the bottom. Genes are displayed as arrows, and variable segments as green boxes.

ing of numerous parameters or to implement a questionable "last chance alignment" phase, making thus possible the *complete automation of high quality, reliable alignment production without further post-processing.* This is an important consideration

when taking into account the increasing number of genomes that will be sequenced in the near future.

Moreover, the results in Section 6.5, illustrate the tendency of ProgressiveMauve to "hatch" the alignment of orthologs (even in a collinear block), while *YOC can align entire syntenic regions*. This should help, *e.g.* to annotate or study the evolution of operons. Interestingly, YOC performs improvements where other tools give lower quality results: on species with highly divergent strains like *B. aphidicola*, *P. marinus* or *L. lactis*.

Conclusion and perspectives

This thesis focused on the *whole genome alignment problem* from two different perspectives: (i) establishing a protocol meant to assess the quality of WGA from both computational and biological points of view and (ii) proposing a novel method for whole genome alignment, YOC, based on two original ideas, the use of local similarities in the first phase of the anchor based strategy together with a novel approach for collinear chaining allowing for overlaps proportional to the lengths of the adjacent fragments.

In the first part of this manuscript (Chapter 3), we presented several results on the WGA problem. First, we defined a protocol for evaluating WGA results, based on computational measures. By using this protocol, we observed that existing tools fail to align pairs of bacterial strains with high levels of divergence. Second, we proposed to improve the *anchor based strategy*, by replacing methods computing short exact (or approximate) matches, with seed and extend methods generating local similarities, in the first phase of the strategy.

In Chapter 5, we defined *a novel collinear chaining problem by allowing proportional overlaps between fragments in the chain*, we formalized this problem, and showed that it can be solved by a dynamic programming algorithm. We thus exhibited the first two dynamic programming algorithms for this problem: one trivial algorithm and one more complex based on the sweep line paradigm. Both algorithms solve the problem in quadratic time in function of the number of fragments. However, we showed that our sweep line algorithm outperforms the truly quadratic dynamic programming solution in practice. Later in this manuscript, in Chapter 6, we also showed that overlaps improve significantly global genome results. In conclusion, our newly introduced definition and algorithms for the problem of collinear chaining with overlaps answer to a biological relevant problem and allow one to improve the results of WGA tools based on the anchor strategy.

Finally, in Chapter 6, we showed that by combining local similarities with an adapted chaining allowing for overlaps, one improves results on divergent strains. The results were obtained on a large dataset of intra-species bacterial genomes containing both collinear and rearranged pairs with different levels of divergence. We introduced *a novel pairwise WGA tool* that we called YOC, which unlike other tools, is composed of only two phases instead of four. Based on our evaluation protocol, we showed that our newly introduced tool has several advantages. Unlike other tools: (i) YOC does not need fine tuning of numerous parameters, (ii) it does not implement a "last chance alignment phase" that is suspected to generate unreliable results, and (iii) it does not need an additional post-processing step. Moreover, by using biological criteria based on orthologous genes, we showed that YOC outputs accurate backbone segmentations

and performs improvements on divergent cases, where other tools obtain lower quality results. In conclusion, YOC makes possible the *complete automation of high quality, reliable alignment production without further post-processing.*

This work opens multiple perspectives. The idea of using local similarities in WGA tools, can be extended to the multiple case. It is a research direction that we pursued in parallel with the work presented in this manuscript, namely an alternative approach to multiple alignment. This alternative approach gave *QOD*, a tool based on a segmentation strategy that may be used in several genome comparison tasks as: identifying novel genes and novel homologous/orthologous genes between several strains, transferring genomic annotations on newly sequenced genomes, comparing unassembled, unfinished genomes, and, maybe most important, identifying specific regions. This method is described in a paper that is under revision in Nucleic Acids Research journal. Thorough analysis of WGA segmentation results and of overlapping regions need to be further pursued. Finally, our pairwise WGA method needs to be extended by allowing rearrangements, in order to apply it to other genomes than those of bacteria.

As for the part of the work related to the chaining problem, comparing proportional overlaps with fixed overlaps, as well as investigating the robustness of chains with respect to the ratio of allowed overlaps, are two points that should be thoroughly studied. Some results on this were included in a paper that is under revision in the Journal of Computational Biology. Moreover, despite the reasonable running times of our program for chaining with overlaps, further work needs to be done in order to compute the average time complexity of the sweep line algorithm. One should also examine different types of graphs instead of the trapezoid graphs, *e.g.* tolerance graphs, that might allow obtaining an improved algorithm for the chaining with overlaps problem. Extending this problem to the multiple case and also to chaining with rearrangements, are necessary research directions. Going from chaining fragments in sequence alignment to chaining seeds in ordered trees and allowing overlaps between such seeds, might also be an interesting idea to pursue (Allali et al., 2011).

From a more general perspective, WGA nowadays needs curated benchmarks, complete user guides for the use of alignment tools, evaluation measures and biological criteria, in order to proceed to systematic alignments. Among other applications described in the manuscript, this should allow, for example, precise orthology assignment. There is also a great need of an up to date review paper, to help the user manage in this mass of tools and WGA studies.

List of Figures

List of Tables

Bibliography

Abouelhoda, M., Kurtz, S., and Ohlebusch, E. (2004). Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86.

Abouelhoda, M. I. and Ohlebusch, E. (2003). A local chaining algorithm and its applications in comparative genomics. In *Proceedings of WABI: International Workshop on Algorithms in Bioinformatics, WABI 2003, LNCS*.

Abouelhoda, M. I. and Ohlebusch, E. (2005). Chaining algorithms for multiple genome comparison. *Journal of Discrete Algorithms*, 3:321–341.

Alexandersson, M., Cawley, S., and Pachter, L. (2003). Slam: cross-species gene finding and alignment with a generalized pair hidden markov model. *Genome Research*, 13(3):496–502.

Allali, J., Chauve, C., Ferraro, P., and Gaillard, A.-L. (2011). Efficient chaining of seeds in ordered trees. In *"Proceedings of the 21st International Conference on Combinatorial Algorithms"*, IWOCA'10, pages 260–273, Berlin, Heidelberg. Springer-Verlag.

Allison, L. and Wallace, C. (1994). The posterior probability distribution of alignments and its application to parameter estimation of evolutionary trees and to optimization of multiple alignments. *Journal of Molecular Evolution*, 39(4):418–430.

Allison, L., Wallace, C., and Yee, C. (1992). Finite-state models in the alignment of macromolecules. *Journal of Molecular Evolution*, 35(1):77–89.

Alm, R., Ling, L., Moir, D., King, B., Brown, E., Doig, P., Smith, D., Noonan, B., Guild, B., et al. (1999). Genomic-sequence comparison of two unrelated isolates of the human gastric pathogen Helicobacter pylori. *Nature*, 397(6715):176–180.

Altschul, S. (1998). Generalized affine gap costs for protein sequence alignment. *Proteins: Structure, Function, and Bioinformatics*, 32(1):88–96.

Altschul, S. and Erickson, B. (1986). Optimal sequence alignment using affine gap costs. *Bulletin of Mathematical Biology*, 48(5):603–616.

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410.

Altschul, S., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389.

Aniba, M., Poch, O., and Thompson, J. (2010). Issues in bioinformatics benchmarking: the case study of multiple sequence alignment. *Nucleic Acids Research*.

Apostolico, A. and Guerra, C. (1987). The longest common subsequence problem revisited. *Algorithmica*, 2:316–336.

Armengol, L., Pujana, M., Cheung, J., Scherer, S., and Estivill, X. (2003). Enrichment of segmental duplications in regions of breaks of synteny between the human and mouse genomes suggest their involvement in evolutionary rearrangements. *Human molecular genetics*, 12(17):2201.

Bafna, V. and Huson, D. H. (2000). The conserved exon method for gene finding. *Proceedings of the International Conference of Intelligent Systems for Molecular Biology*, 8:3–12.

Bafna, V., Narayanan, B. O., and Ravi, R. (1996). Nonoverlapping local alignments (weighted independent sets of axis-parallel rectangles). *Discrete Applied Mathematics*, 71(1-3):41–53.

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

Baldi, P., Chauvin, Y., Hunkapiller, T., and McCLure, M. (1993). Hidden Markov models in molecular biology: new algorithms and applications. *Advances in Neural Information Processing Systems*, pages 747–754.

Baldi, P., Chauvin, Y., Hunkapiller, T., and McClure, M. (1994). Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences*, 91(3):1059.

Balding, D., Bishop, M., and Cannings, C. (2003). *Handbook of statistical genetics*.

Barton, G. and Sternberg, M. (1987). Evaluation and improvements in the automatic alignment of protein sequences. *Protein Engineering Design and Selection*, 1(2):89.

Batzoglou, S., Pachter, L., Mesirov, J., Berger, B., and Lander, E. (2000). Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Research*, 10(7):950.

Bérard, S., Chateau, A., Chauve, C., Paul, C., and Tannier, E. (2009). Computation of perfect DCJ rearrangement scenarios with linear and circular chromosomes. *Journal of Computational Biology*, 16(10):1287–1309.

Bérard, S. and Rivals, E. (2003). Comparison of minisatellites. *Journal of Computational Biology*, 10(3-4):357–372.

Bigot, S., Saleh, O., Lesterlin, C., Pages, C., El Karoui, M., Dennis, C., Grigoriev, M., Allemand, J., Barre, F., and Cornet, F. (2005). KOPS: DNA motifs that control E. coli chromosome segregation by orienting the FtsK translocase. *The EMBO journal*, 24(21):3770–3780.

Blackshields, G., Wallace, I., Larkin, M., and Higgins, D. (2006). Analysis and comparison of benchmarks for multiple sequence alignment. *In silico Biology*, 6(4):321–339.

Blanchette, M. (2007). Computation and analysis of genomic multi-sequence alignments. *Annual Review of Genomics and Human Genetics*, 8(1):193.

Blanchette, M., Diallo, A., Green, E., Miller, W., and Haussler, D. (2008). Computational reconstruction of ancestral DNA sequences. *Methods in Molecular Biology*, 422:171.

Blanchette, M., Kent, W., Riemer, C., Elnitski, L., Smit, A., Roskin, K., Baertsch, R., Rosenbloom, K., Clawson, H., Green, E., et al. (2004). Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research*, 14(4):708.

Bourque, G. and Tesler, G. (2008). Computational tools for the analysis of rearrangements in mammalian genomes. *Methods in molecular biology (Clifton, NJ)*, 452:431.

Bourque, G., Zdobnov, E., Bork, P., Pevzner, P., and Tesler, G. (2005). Comparative architectures of mammalian and chicken genomes reveal highly variable rates of genomic rearrangements across different lineages. *Genome Research*, 15(1):98.

Bray, N., Dubchak, I., and Pachter, L. (2003). Avid: A global alignment program. *Genome Research*, 13(1):97–102.

Bray, N. and Pachter, L. (2003). MAVID multiple alignment server. *Nucleic acids research*, 31(13):3525.

Brenner, S., Chothia, C., and Hubbard, T. (1998). Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proceedings of the National Academy of Sciences*, 95(11):6073.

Brudno, M., Chapman, M., Göttgens, B., Batzoglou, S., and Morgenstern, B. (2003a). Fast and sensitive multiple alignment of large genomic sequences. *BMC Bioinformatics*, 4(1):66.

Brudno, M., Do, C. B., Cooper, G. M., Kim, M. F., Davydov, E., Green, E. D., Sidow, A., and Batzoglou, S. (2003b). Lagan and multi-lagan: efficient tools for large-scale multiple alignment of genomic dna. *Genome Research*, 13(4):721–731.

Brudno, M., Malde, S., Poliakov, A., Do, C. B., Couronne, O., Dubchak, I., and Batzoglou, S. (2003c). Glocal alignment: finding rearrangements during alignment. *Bioinformatics*, 19(S1):i54–62.

Burge, C. and Karlin, S. (1997). Prediction of complete gene structures in human genomic DNA1. *Journal of Molecular Biology*, 268(1):78–94.

Cai, W., Pei, J., and Grishin, N. V. (2004). Reconstruction of ancestral protein sequences and its applications. *BMC Evolutionary Biology*, 4(1):33+.

Calabrese, P. P., Chakravarty, S., and Vision, T. J. (2003). Fast identification and statistical evaluation of segmental homologies in comparative maps. In *ISMB (Supplement of Bioinformatics): Proceedings of the Eleventh International Conference on Intelligent Systems for Molecular Biology, June 29 - July 3, 2003, Brisbane, Australia*, pages 74–80.

Carrillo, H. and Lipman, D. (1988). The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*, 48(5):1073–1082.

Carroll, H., Beckstead, W., O'Connor, T., Ebbert, M., Clement, M., Snell, Q., and McClellan, D. (2007). DNA reference alignment benchmarks based on tertiary structure of encoded proteins. *Bioinformatics*, 23(19):2648.

Carroll, S., Prud'homme, B., and Gompel, N. (2008). Regulating evolution. *Scientific American Magazine*, 298(5):60–67.

Carter, D. and Durbin, R. (2006). Vertebrate gene finding from multiple-species alignments using a two-level strategy. *Genome Biology*, 7 Suppl 1.

Cartwright, R. (2005). DNA assembly with gaps (Dawg): simulating sequence evolution. *Bioinformatics*, 21(Suppl 3).

Chain, P., Kurtz, S., Ohlebusch, E., and Slezak, T. (2003). An applications-focused review of comparative genomics tools: Capabilities, limitations and future challenges. *Briefings in Bioinformatics*, 4(2):105.

Chang, M. and Benner, S. (2004). Empirical analysis of protein insertions and deletions determining parameters for the correct placement of gaps in protein sequence alignments. *Journal of Molecular Biology*, 341(2):617–631.

Chao, K., Pearson, W., and Miller, W. (1992). Aligning two sequences within a specified diagonal band. *Bioinformatics*, 8(5):481.

Chao, K., Zhang, J., Ostell, J., and Miller, W. (1995). A local alignment tool for very long DNA sequences. *Bioinformatics*, 11(2):147.

Chao, K., Zhang, J., Ostell, J., and Miller, W. (1997). A tool for aligning very similar DNA sequences. *Bioinformatics*, 13(1):75.

Chapman, M., Charchar, F., Kinston, S., Bird, C., Grafham, D., Rogers, J., Grützner, F., Marshall Graves, J., Green, A., and Göttgens, B. (2003). Comparative and functional analyses of LYL1 loci establish marsupial sequences as a model for phylogenetic footprinting. *Genomics*, 81(3):249–259.

Chen, X. and Tompa, M. (2010). Comparative assessment of methods for aligning multiple genome sequences. *Nature Biotechnology*, 28(6):567–572.

Chiapello, H., Bourgait, I., Sourivong, F., Heuclin, G., Gendrault-Jacquemard, A., Petit, M., and El Karoui, M. (2005). Systematic determination of the mosaic structure of bacterial genomes: species backbone versus strain-specific loops. *BMC Bioinformatics*, 6(1):171.

Chiapello, H., Gendrault, A., Caron, C., Blum, J., Petit, M., and El Karoui, M. (2008). MOSAIC: an online database dedicated to the comparative genomics of bacterial strains at the intra-species level. *BMC Bioinformatics*, 9(1):498.

Claire, L. and Sagot, M. (2008). A small trip in the untranquil world of genomes A survey on the detection and analysis of genome rearrangement breakpoints. *Theoretical Computer Science*, 395(2-3):171–192.

Cobb, J., Büsst, C., Petrou, S., Harrap, S., and Ellis, J. (2008). Searching for functional genetic variants in non-coding DNA. *Clinical and Experimental Pharmacology and Physiology*, 35(4):372–375.

Collins, F., Green, E., Guttmacher, A., and Guyer, M. (2003). A vision for the future of genomics research. *Nature*, 422(6934):835–847.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to algorithms*. MIT Press, 2nd edition.

Corpet, F. (1988). Multiple sequence alignment with hierarchical clustering. *Nucleic Acids Research*, 16(22):10881.

Coventry, A., Kleitman, D. J., and Berger, B. (2004). Msari: multiple sequence alignments for statistical detection of rna secondary structure. *Proceedings of the National Academy of Sciences*, 101(33):12102–12107.

Dagan, I., Golumbic, M. C., and Pinter, R. Y. (1988). Trapezoid graphs and their coloring. *Discrete Applied Mathematics*, 21(1):35–46.

Darling, A., Mau, B., and Perna, N. (2010). progressiveMauve: Multiple Genome Alignment with Gene Gain, Loss and Rearrangement. *PloS one*, 5(6):e11147.

Darling, A., Treangen, T., Zhang, L., Kuiken, C., Messeguer, X., and Perna, N. (2006). Procrastination leads to efficient filtration for local multiple alignment. *Algorithms in Bioinformatics*, pages 126–137.

Darling, A. C., Mau, B., Blattner, F. R., and Nicole T. Perna (2004). Mauve: Multiple Alignment of Conserved Genomic Sequence With Rearrangements. *Genome Research*, 14(7):1394–1403.

Dayhoff, M., Schwartz, R., and Orcutt, B. (1978). A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5(Suppl 3):345–352.

De Berg, M., Cheong, O., Van Kreveld, M., and Overmars, M. (2008). *Computational geometry: algorithms and applications*. Springer-Verlag New York Inc.

De Bruijn, N. (1969). A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 6:33–40.

Delcher, A. L., Kasif, S., Fleischmann, R. D., Peterson, J., White, O., and Salzberg, S. L. (1999). Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376.

Delcher, A. L., Phillippy, A., Carlton, J., and Salzberg, S. L. (2002). Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research*, 30(11):2478–2483.

Delsuc, F., Brinkmann, H., and Philippe, H. (2005). Phylogenomics and the reconstruction of the tree of life. *Nature Reviews Genetics*, 6(5):361–375.

Deogun, J., Yang, J., and Ma, F. (2004). Emagen: An efficient approach to multiple whole genome alignment. In *Proceedings of the second Conference on Asia-Pacific bioinformatics-Volume 29*, page 122. Australian Computer Society, Inc.

Devillers, H., Chiapello, H., Schbath, S., and Karoui, M. E. (2010). Assessing the robustness of complete bacterial genome segmentations. In *Proceedings of Comparative Genomics, RECOMB 2010 International Workshop, RCG 2010, Ottawa, Canada, October 9-11, 2010*, Lecture Notes in Computer Science, pages 173–187.

Di Bernardo, D., Down, T., and Hubbard, T. (2003). ddbrna: detection of conserved secondary structures in multiple alignments. *Bioinformatics*, 19(13):1606–1611.

Do, C., Mahabhashyam, M., Brudno, M., and Batzoglou, S. (2005). ProbCons: probabilistic consistency-based multiple sequence alignment. *Genome Research*, 15(2):330.

Doolittle, R. (1995). The multiplicity of domains in proteins. *Annual Review of Biochemistry*, 64(1):287–314.

Dowell, R. and Eddy, S. (2004). Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5(1):71.

Dubchak, I., Brudno, M., Loots, G. G., Pachter, L., Mayor, C., Rubin, E. M., and Frazer, K. A. (2000). Active conservation of noncoding sequences revealed by three-way species comparisons. *Genome Research*, 10(9):1304–1306.

Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis*. Cambridge university press Cambridge, UK.

Eddy, S. (1995). Multiple alignment using hidden Markov models. In *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, volume 3, pages 114–120.

Edgar, R. (2004a). MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(1):113.

Edgar, R. (2004b). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792.

Eichler, E. and Sankoff, D. (2003). Structural dynamics of eukaryotic chromosome evolution. *Science*, 301(5634):793.

Eppstein, D., Galil, Z., Giancarlo, R., and Italiano, G. F. (1992a). Sparse dynamic programming i: Linear cost functions. *Journal of Association for Computing Machinery*, 39(3):519–545.

Eppstein, D., Galil, Z., Giancarlo, R., and Italiano, G. F. (1992b). Sparse dynamic programming ii: Convex and concave cost functions. *Journal of Association for Computing Machinery*, 39(3):546–567.

Ewan Birney, J., Anindya Dutta, R., Thomas, R., Elliott, H., Zhiping Weng, M., Emmanouil, T., John, A., Robert, E., Michael, S., Christopher, M., et al. (2007). Identification and analysis of functional elements in 1% of the human genome by the ENCODE pilot project. *Nature*, 447(7146):799–816.

Felsenstein, J. (1981). Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376.

Felsenstein, J. (2004). *Inferring phylogenies*. Sinauer Associates, Sunderland, Mass.

Felsner, S., Muller, R., and Wernisch, L. (1995). Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74:13–32.

Feng, D. and Doolittle, R. (1987). Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25(4):351–360.

Fitch, J., Gardner, S., Kuczmarski, T., Kurtz, S., Myers, R., Ott, L., Slezak, T., Vitalis, E., Zemla, A., and McCready, P. (2002). Rapid development of nucleic acid diagnostics. *Proceedings of the IEEE*, 90(11):1708–1721.

Fitch, W. (1966). An improved method of testing for evolutionary homology. *Journal of Molecular Biology*, 16(1):9–16.

Flicek, P., Keibler, E., Hu, P., Korf, I., and Brent, M. R. (2003). Leveraging the mouse genome for gene prediction in human: from whole-genome shotgun reads to a global synteny map. *Genome Research*, 13(1):46–54.

Friedberg, R., Darling, A. E., and Yancopoulos, S. (2008). Genome rearrangement by the double cut and join operation. *Methods in molecular biology*, 452:385–416.

Frith, M., Hamada, M., and Horton, P. (2010). Parameters for accurate genome alignment. *BMC bioinformatics*, 11(1):80.

Frutos, R., Viari, A., Ferraz, C., Morgat, A., Eychenie, S., Kandassamy, Y., Chantal, I., Bensaid, A., Coissac, E., Vachiery, N., et al. (2006). Comparative genomic analysis of three strains of Ehrlichia ruminantium reveals an active process of genome size plasticity. *Journal of Bacteriology*, 188(7):2533.

Gardner, P., Wilm, A., and Washietl, S. (2005). A benchmark of multiple sequence alignment programs upon structural RNAs. *Nucleic Acids Research*, 33(8):2433.

Gascuel, O. (2005). *Mathematics of evolution and phylogeny*. Oxford University Press, USA.

Gelfand, M. S., Mironov, A., and Pevzner, P. (1996). Gene recognition via spliced alignment. In *Proceedings of the National Academy of Sciences*, volume 93, pages 9061–9066.

Golumbic, M. C. (1980). *Algorithmic graph theory and perfect graphs*. Academic Press, Inc., San Diego.

Good, I. (1946). Normal recurring decimals. *Journal of the London Mathematical Society*, 1(3):167.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708.

Gotoh, O. (1986). Alignment of three biological sequences with an efficient traceback procedure. *Journal of Theoretical Biology*, 121(3):327–337.

Gotoh, O. (1999). Multiple sequence alignment: algorithms and applications. *Advances in Biophysics*, 36(1):159–206.

Göttgens, B., Barton, L., Gilbert, J., Bench, A., Sanchez, M., Bahn, S., Mistry, S., Grafham, D., McMurray, A., Vaudin, M., et al. (2000). Analysis of vertebrate SCL loci identifies conserved enhancers. *Nature Biotechnology*, 18(2):181–186.

Gusfield, D. (1997). *Algorithms on strings, trees and sequences*. Cambridge University Press.

Hacker, J. and Carniel, E. (2001). Ecological fitness, genomic islands and bacterial pathogenicity. *EMBO reports*, 2(5):376–381.

Hall, B. (2008). Simulating DNA coding sequence evolution with EvolveAGene 3. *Molecular Biology and Evolution*, 25(4):688.

Hall, B. G. (2006). Simple and accurate estimation of ancestral protein sequences. *Proceedings of the National Academy of Sciences*, 103(14):5431–5436.

Halpern, D., Chiapello, H., Schbath, S., Robin, S., Hennequet-Antier, C., Gruss, A., and El Karoui, M. (2007). Identification of DNA motifs implicated in maintenance of bacterial core genomes by predictive modeling. *PLoS Genetics*, 3(9):1614–1621.

Hardison, R. C. (2003). Comparative genomics. *PLoS Biology*, 1(2):e58.

Hasegawa, M., Kishino, H., and Yano, T. (1985). Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, 22(2):160–174.

Hayashi, T., Makino, K., Ohnishi, M., Kurokawa, K., Ishii, K., Yokoyama, K., Han, C., Ohtsubo, E., Nakayama, K., Murata, T., et al. (2001). Complete genome sequence of enterohemorrhagic Eschelichia coli O157: H7 and genomic comparison with a laboratory strain K-12. *DNA research*, 8(1):11.

Hein, J. (1989). A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. *Molecular Biology and Evolution*, 6(6):649.

Henikoff, S. and Henikoff, J. (1992). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915.

Hess, J., Fox, M., Schmid, C., and Shen, C. (1983). Molecular evolution of the human adult alpha-globin-like gene region: insertion and deletion of Alu family repeats and non-Alu DNA sequences. *Proceedings of the National Academy of Sciences of the United States of America*, 80(19):5970.

Higgins, D. and Sharp, P. (1988). CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244.

Higgins, D. and Taylor, W. (2000). *Bioinformatics: sequence, structure, and databanks*. Oxford University Press.

Hirschberg, D. S. (1977). Algorithms for the longest common subsequence problem. *Journal of Association for Computing Machinery*, 24(4):664–675.

Hogeweg, P. and Hesper, B. (1984). The alignment of sets of sequences and the construction of phyletic trees: an integrated method. *Journal of Molecular Evolution*, 20(2):175–186.

Hohl, M., Kurtz, S., and Ohlebusch, E. (2002). Efficient multiple genome alignment. *Bioinformatics*, 18(S1):S312–320.

Holmes, I. (2005). Accelerated probabilistic inference of RNA structure evolution. *BMC Bioinformatics*, 6(1):73.

Huang, W., Umbach, D., and Li, L. (2005). Accurate anchoring alignment of divergent sequences. *Bioinformatics*, 22(1):29.

Huang, X., Hardison, R., and Miller, W. (1990). A space-efficient algorithm for local similarities. *Bioinformatics*, 6(4):373.

Huang, X. and Miller, W. (1991). A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics*, 12(3):337–357.

Hughey, R. and Krogh, A. (1996). Hidden Markov models for sequence analysis: extension and analysis of the basic method. *Bioinformatics*, 12(2):95.

Jackson, A., Gamble, J., Yeomans, T., Moran, G., Saunders, D., Harris, D., Aslett, M., Barrell, J., Butler, G., Citiulo, F., et al. (2009). Comparative genomics of the fungal pathogens Candida dubliniensis and Candida albicans. *Genome Research*, 19(12):2231.

Jacobson, G. and Vo, K.-P. (1992). Heaviest increasing/common subsequence problems. In *Proceedings of Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson, Arizona, USA, April 29 - May 1, 1992*, volume 644, pages 52–66.

Johnson, D. B. (1982). A priority queue in which initialization and queue operations take O(log log d) time. *Mathematical Systems Theory*, 15(4):295–309.

Jones, D., Taylor, W., and Thornton, J. (1992). The rapid generation of mutation data matrices from protein sequences. *Bioinformatics*, 8(3):275.

Joseph, D., Meidanis, J., and Tiwari, P. (1992). Determining dna sequence similarity using maximum independent set algorithms for interval graphs. In *Proceedings of Algorithm Theory - SWAT '92, Third Scandinavian Workshop on Algorithm Theory, Helsinki, Finland, July 8-10, 1992*, volume 621 of *Lecture Notes in Computer Science*, pages 326–337.

Jue, R., Woodbury, N., and Doolittle, R. (1980). Sequence homologies among E. coli ribosomal proteins: Evidence for evolutionarily related groupings and internal duplications. *Journal of Molecular Evolution*, 15(2):129–148.

Jukes, T. and Cantor, C. (1969). Evolution of protein molecules. *Mammalian Protein Metabolism*, 3:21–132.

Kann, M., Qian, B., and Goldstein, R. (2000). Optimization of a new score function for the detection of remote homologs. *Proteins: Structure, Function, and Bioinformatics*, 41(4):498–503.

Karlin, S. and Altschul, S. (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87(6):2264.

Kasai, T., Lee, G., Arimura, H., Arikawa, S., and Park, K. (2001). Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Combinatorial Pattern Matching*, pages 181–192. Springer.

Katoh, K., Kuma, K., Toh, H., and Miyata, T. (2005). MAFFT version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Research*, 33(2):511.

Katoh, K., Misawa, K., Kuma, K., and Miyata, T. (2002). MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, 30(14):3059.

Kellis, M., Patterson, N., Birren, B., Berger, B., and Lander, E. (2004). Methods in comparative genomics: genome correspondence, gene identification and regulatory motif discovery. *Journal of Computational Biology*, 11(2-3):319–355.

Kellis, M., Patterson, N., Endrizzi, M., Birren, B., and Lander, E. (2003). Sequencing and comparison of yeast species to identify genes and regulatory elements. *Nature*, 423(6937):241–254.

Kent, W., Baertsch, R., Hinrichs, A., Miller, W., and Haussler, D. (2003). Evolution's cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. *Proceedings of the National Academy of Sciences of the United States of America*, 100(20):11484.

Kent, W. and Zahler, A. (2000). Conservation, regulation, synteny, and introns in a large-scale C. briggsae–C. elegans genomic alignment. *Genome Research*, 10(8):1115.

Kimura, M. (1980). A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16(2):111–120.

Kirkness, E. F., Bafna, V., Halpern, A. L., Levy, S., Remington, K., Rusch, D. B., Delcher, A. L., Pop, M., Wang, W., Fraser, C. M., and Venter, J. C. (2003). The dog genome: survey sequencing and comparative analysis. *Science*, 301(5641):1898–1903.

Knudsen, B. and Hein, J. (1999). RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, 15(6):446.

Knuth, D. (1998). *The art of computer programming*, volume 3 / Sorting and Searching. Addison-Wesley.

Kolchanov, N., Ignatieva, E., Ananko, E., Podkolodnaya, O., Stepanenko, I., Merkulova, T., Pozdnyakov, M., Podkolodny, N., Naumochkin, A., and Romashchenko, A. (2002). Transcription regulatory regions database (TRRD): its status in 2002. *Nucleic Acids Research*, 30(1):312.

Konstantinidis, K. and Tiedje, J. (2005). Genomic insights that advance the species definition for prokaryotes. *Proceedings of the National Academy of Sciences of the United States of America*, 102(7):2567.

Korf, I., Flicek, P., Duan, D., and Brent, M. R. (2001). Integrating genomic homology into gene structure prediction. *Bioinformatics*, 17 Suppl 1.

Krane, D. and Raymer, M. (2003). *Fundamental concepts of bioinformatics*. Pearson Education India.

Krogh, A. and Brown, I. (1994). Hidden Markov models in computational biology. *Journal of Molecular Biology*, 235:1501–1531.

Kucherov, G., Noé, L., and Roytberg, M. (2006). A unifying framework for seed sensitivity and its application to subset seeds. *Journal of Bioinformatics and Computational Biology*, 4(2):553.

Kucherov, G., Noé, L., and Roytberg, M. (2007). Subset seed automaton. In *Proceedings of the 12th international conference on Implementation and application of automata*, pages 180–191. Springer-Verlag.

Kumar, S. (1996). A stepwise algorithm for finding minimum evolution trees. *Molecular Biology and Evolution*, 13(4):584–593.

Kurtz, S. (2003). The Vmatch large scale sequence analysis software. *Ref Type: Computer Program*, pages 4–12.

Kurtz, S., Choudhuri, J., Ohlebusch, E., Schleiermacher, C., Stoye, J., and Giegerich, R. (2001). REPuter: the manifold applications of repeat analysis on a genomic scale. *Nucleic Acids Research*, 29(22):4633.

Kurtz, S., Phillippy, A., Delcher, A., Smoot, M., Shumway, M., Antonescu, C., and Salzberg, S. (2004). Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):r12.

Lander, E., Linton, L., Birren, B., Nusbaum, C., Zody, M., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., et al. (2001). Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921.

Lassmann, T. and Sonnhammer, E. (2002). Quality assessment of multiple alignment programs. *FEBS letters*, 529(1):126–130.

Leclercq, S., Rivals, E., and Jarne, P. (2010). DNA slippage occurs at microsatellite loci without minimal threshold length in humans: a comparative genomic approach. *Genome Biology and Evolution*, 2(0):325.

Lee, C., Grasso, C., and Sharlow, M. (2002). Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452.

Lefébure, T. and Stanhope, M. (2009). Pervasive, genome-wide positive selection leading to functional divergence in the bacterial genus Campylobacter. *Genome Research*, 19(7):1224.

Lemaitre, C., Tannier, E., Gautier, C., and Sagot, M. (2008). Precise detection of rearrangement breakpoints in mammalian chromosomes. *BMC Bioinformatics*, 9(1):286.

Li, W. and Olmstead, R. (1997). *Molecular evolution*. Sinauer Associates Sunderland, MA.

Li, X. and Waterman, M. (2003). Estimating the Repeat Structure and Length of DNA Sequences Using l-Tuples. *Genome Research*, 13(8):1916.

Lipman, D. and Pearson, W. (1985). Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435.

Lippert, R. A., Zhao, X., Florea, L., Mobarry, C., and Istrail, S. (2004). Finding anchors for genomic sequence comparison. In *RECOMB '04: Proceedings of the eighth annual international conference on Research in Computational Molecular Biology*, pages 233–241, New York, NY, USA. ACM Press.

Lunter, G., Rocco, A., Mimouni, N., Heger, A., Caldeira, A., and Hein, J. (2008). Uncertainty in homology inferences: assessing and improving genomic sequence alignment. *Genome Research*, 18(2):298.

Ma, B., Tromp, J., and Li, M. (2002). PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440.

Maione, D., Margarit, I., Rinaudo, C., Masignani, V., Mora, M., Scarselli, M., Tettelin, H., Brettoni, C., Iacobini, E., Rosini, R., et al. (2005). Identification of a universal Group B streptococcus vaccine by multiple genome screen. *Science*, 309(5731):148.

Margulies, E., Cooper, G., Asimenos, G., Thomas, D., Dewey, C., Siepel, A., Birney, E., Keefe, D., Schwartz, A., Hou, M., et al. (2007). Analyses of deep mammalian sequence alignments and constraint predictions for 1% of the human genome. *Genome Research*, 17(6):760.

McClelland, M., Florea, L., Sanderson, K., Clifton, S., Parkhill, J., Churcher, C., Dougan, G., Wilson, R., and Miller, W. (2000). Comparison of the Escherichia coli K-12 genome with sampled genomes of a Klebsiella pneumoniae and three Salmonella enterica serovars, Typhimurium, Typhi and Paratyphi. *Nucleic Acids Research*, 28(24):4974.

McClure, M., Vasi, T., and Fitch, W. (1994). Comparative analysis of multiple protein-sequence alignment methods. *Molecular Biology and Evolution*, 11(4):571–592.

McCreight, E. (1976). A space-economical suffix tree construction algorithm. *Journal of Association for Computing Machinery*, 23(2):262–272.

Metzker, M. (2009). Sequencing technologies—the next generation. *Nature Reviews Genetics*, 11(1):31–46.

Miller, M. P. and Kumar, S. (2001). Understanding human disease mutations through the use of interspecific genetic variation. *Human Molecular Genetics*, 10(21):2319–2328.

Miller, M. P., Parker, J. D., Rissing, S. W., and Kumar, S. (2003). Quantifying the intragenic distribution of human disease mutations. *Annals of Human Genetics*, 67(Pt 6):567–579.

Miller, W. (2001). Comparison of genomic DNA sequences: solved and unsolved problems. *Bioinformatics*, 17(5):391.

Miller, W., Makova, K. D., Nekrutenko, A., and Hardison, R. C. (2004). Comparative genomics. *Annual Review of Genomics and Human Genetics*, 5(1):15–56.

Mizuguchi, K., Deane, C., Blundell, T., and Overington, J. (1998). HOMSTRAD: a database of protein structure alignments for homologous families. *Protein Science*, 7(11):2469–2471.

Morgenstern, B. (2002). A simple and space-efficient fragment-chaining algorithm for alignment of dna and protein sequences. *Applied Mathematics Letters*, 15(1):11–16.

Morgenstern, B., Frech, K., Dress, A. W. M., and Werner, T. (1998). Dialign: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294.

Mott, R. (1992). Maximum-likelihood estimation of the statistical distribution of Smith-Waterman local sequence similarity scores. *Bulletin of Mathematical Biology*, 54(1):59–75.

Mott, R. (1997). EST_GENOME: a program to align spliced DNA sequences to unspliced genomic DNA. *Computer Applications in the Biosciences*, 13(4):477–478.

Mott, R. (1999). Local sequence alignments with monotonic gap penalties. *Bioinformatics*, 15(6):455.

Murata, M., Richardson, J., and Sussman, J. (1985). Simultaneous comparison of three protein sequences. *Proceedings of the National Academy of Sciences*, 82(10):3073.

Myers, E. and Miller, W. (1988). Optimal alignments in linear space. *Bioinformatics*, 4(1):11.

Myers, G. and Miller, W. (1995). Chaining multiple-alignment fragments in sub-quadratic time. In *Proceedings of the sixth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 38–47.

Needleman, S. and Blair, T. (1969). Homology of Pseudomonas cytochrome c-551 with eukaryotic c-cytochromes. *Proceedings of the National Academy of Sciences*, 63(4):1227.

Needleman, S. and Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453.

Noé, L. and Kucherov, G. (2005). YASS: enhancing the sensitivity of DNA similarity search. *Nucleic Acids Research*, 33(Web Server Issue):W540.

Notredame, C. (2007). Recent evolutions of multiple sequence alignment algorithms. *PLoS Computational Biology*, 3(8).

Notredame, C., Higgins, D., and Heringa, J. (2000). T-coffee: a novel method for fast and accurate multiple sequence alignment1. *Journal of Molecular Biology*, 302(1):205–217.

Ogasawara, J. and Morishita, S. (2003). A fast and sensitive algorithm for aligning ests to the human genome. *Journal of Bioinformatics and Computational Biology*, 1(2):363–386.

Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H., and Kanehisa, M. (1999). KEGG: Kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 27(1):29.

O'Sullivan, O., Suhre, K., Abergel, C., Higgins, D., and Notredame, C. (2004). 3DCoffee: combining protein sequences and structures within multiple sequence alignments. *Journal of Molecular Biology*, 340(2):385–395.

Paten, B., Herrero, J., Beal, K., Fitzgerald, S., and Birney, E. (2008). Enredo and Pecan: Genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome Research*, 18(11):1814.

Pearson, W. and Lipman, D. (1988). Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444.

Pedersen, J. S., Bejerano, G., Siepel, A., Rosenbloom, K., Lindblad-Toh, K., Lander, E. S., Kent, J., Miller, W., and Haussler, D. (2006). Identification and classification of conserved rna secondary structures in the human genome. *PLoS Computational Biology*, 2(4):e33+.

Perna, N., Plunkett, G., Burland, V., Mau, B., Glasner, J., Rose, D., Mayhew, G., Evans, P., Gregor, J., Kirkpatrick, H., et al. (2001). Genome sequence of enterohaemorrhagic Escherichia coli O157: H7. *Nature*, 409(6819):529–533.

Pevzner, P., Tang, H., and Tesler, G. (2004). De novo repeat classification and fragment assembly. *Genome Research*, 14(9):1786.

Pevzner, P., Tang, H., and Waterman, M. (2001). An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America*, 98(17):9748.

Pevzner, P. and Tesler, G. (2003). Genome rearrangements in mammalian evolution: Lessons from human and mouse genomes. *Genome Research*, 13(1):37–45.

Pollard, D., Bergman, C., Stoye, J., Celniker, S., and Eisen, M. (2004). Benchmarking tools for the alignment of functional noncoding DNA. *BMC Bioinformatics*, 5(1):6.

Prakash, A. and Tompa, M. (2007). Measuring the accuracy of genome-size multiple alignments. *Genome Biology*, 8(6):R124.

Rabiner, L. and Juang, B. (1986). An introduction to hidden Markov models. *IEEE ASSp Magazine*, 3(1 Part 1):4–16.

Raghava, G., Searle, S., Audley, P., Barber, J., and Barton, G. (2003). OXBench: a benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics*, 4(1):47.

Rambaut, A. and Grass, N. (1997). Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Bioinformatics*, 13(3):235.

Raphael, B., Zhi, D., Tang, H., and Pevzner, P. (2004). A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Research*, 14(11):2336.

Reese, J. and Pearson, W. (2002). Empirical determination of effective gap penalties for sequence comparison. *Bioinformatics*, 18(11):1500.

Rivas, E. and Eddy, S. R. (2001). Noncoding rna gene detection using comparative sequence analysis. *BMC Bioinformatics*, 2(1):8+.

Rosenberg, M. (2005a). Evolutionary distance estimation and fidelity of pair wise sequence alignment. *BMC Bioinformatics*, 6(1):102.

Rosenberg, M. (2005b). MySSP: Non-stationary evolutionary sequence simulation, including indels. *Evolutionary Bioinformatics Online*, 1:81.

Rosenberg, M. (2009). *Sequence alignment: methods, models, concepts, and strategies*. University of California Press.

Roytberg, M., Ogurtsov, A., Shabalina, S., and Kondrashov, A. (2002). A hierarchical approach to aligning collinear regions of genomes. *Bioinformatics*, 18(12):1673.

Rubin, G., Yandell, M., Wortman, J., Gabor Miklos, G., Nelson, C., Hariharan, I., Fortini, M., Li, P., Apweiler, R., Fleischmann, W., et al. (2000). Comparative genomics of the eukaryotes. *Science*, 287(5461):2204.

Saitou, N. and Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406.

Sankoff, D. and Blanchette, M. (1998). Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5(3):555–570.

Sankoff, D., Cedergren, R., and Lapalme, G. (1976). Frequency of insertion-deletion, transversion, and transition in the evolution of 5S ribosomal RNA. *Journal of Molecular Evolution*, 7(2):133–149.

Sankoff, D. and Nadeau, J. (2000). *Comparative genomics*. Kluwer Academic.

Schneider, A., Dessimoz, C., and Gonnet, G. (2007). OMA Browser–exploring orthologous relations across 352 complete genomes. *Bioinformatics*, 23(16):2180.

Schneider, D., Duperchy, E., Depeyrot, J., Coursange, E., Lenski, R., and Blot, M. (2002). Genomic comparisons among Escherichia coli strains B, K-12, and O157: H7 using IS elements as molecular markers. *BMC Microbiology*, 2:18.

Schuster, S. (2007). Next-generation sequencing transforms today's biology. *Nature Methods*, 5(1):16–18.

Schwartz, S., Kent, W., Smit, A., Zhang, Z., Baertsch, R., Hardison, R., Haussler, D., and Miller, W. (2003). Human–mouse alignments with BLASTZ. *Genome Research*, 13(1):103.

Schwartz, S., Zhang, Z., Frazer, K., Smit, A., Riemer, C., Bouck, J., Gibbs, R., Hardison, R., and Miller, W. (2000). PipMaker—a web server for aligning two genomic DNA sequences. *Genome Research*, 10(4):577.

Sellers, P. (1980). The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, 1(4):359–373.

Serruto, D. and Rappuoli, R. (2006). Post-genomic vaccine development. *FEBS letters*, 580(12):2985–2992.

Serruto, D., Serino, L., Masignani, V., and Pizza, M. (2009). Genome-based approaches to develop vaccines against bacterial pathogens. *Vaccine*, 27(25-26):3245–3250.

Shibuya, T. and Kurochkin, I. (2003). Match chaining algorithms for cDNA mapping. In *Proceedings of WABI: International Workshop on Algorithms in Bioinformatics, WABI 2003, LNCS*, volume 2812, pages 462–475.

Sim, S., Easterbrook, S., and Holt, R. (2003). Using benchmarking to advance research: A challenge to software engineering. In *Proceedings of the 25th International Conference on Software Engineering*, page 83. IEEE Computer Society.

Smith, T. and Waterman, M. (1981). Comparison of biosequences. *Advances in Applied Mathematics*, 2(4):482–489.

Sobel, E. and Martinez, H. M. (1986). A multiple sequence alignment program. *Nucleic Acids Research*, 14(1):363–374.

Stanke, M., Schöffmann, O., Morgenstern, B., and Waack, S. (2006). Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. *BMC Bioinformatics*, 7(1):62.

Sterk, P., Kersey, P., and Apweiler, R. (2006). Genome Reviews: Standardizing Content and Representation of Information about Complete Genomes. *OMICS*, 10(2):114–118.

Stojanovic, N., Florea, L., Riemer, C., Gumucio, D., Slightom, J., Goodman, M., Miller, W., and Hardison, R. (1999). Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions. *Nucleic Acids Research*, 27(19):3899.

Stoye, J., Evers, D., and Meyer, F. (1998). Rose: generating sequence families. *Bioinformatics*, 14(2):157.

Strope, C., Scott, S., and Moriyama, E. (2007). Indel-Seq-Gen: a new protein family simulator incorporating domains, motifs, and indels. *Molecular Biology and Evolution*, 24(3):640.

Subirana, J. and Messeguer, X. (2010). The most frequent short sequences in noncoding DNA. *Nucleic Acids Research*, 38(4):1172.

Sundararajan, M., Brudno, M., Small, K., Sidow, A., and Batzoglou, S. (2004). Chaining algorithms for alignment of draft sequence. In *Proceedings of WABI: International Workshop on Algorithms in Bioinformatics, WABI 2004, LNCS*.

Swidan, F., Rocha, E., Shmoish, M., and Pinter, R. (2006). An integrative method for accurate comparative genome mapping. *PLoS Computational Biology*, 2(8):e75.

Swidan, F. and Shamir, R. (2009). Assessing the Quality of Whole Genome Alignments in Bacteria. *Advances in Bioinformatics*, 2009.

Takahashi, K. and Nei, M. (2000). Efficiencies of fast algorithms of phylogenetic inference under the criteria of maximum parsimony, minimum evolution, and maximum likelihood when a large number of sequences are used. *Molecular Biology and Evolution*, 17(8):1251–1258.

Tatusov, R., Galperin, M., Natale, D., and Koonin, E. (2000). The COG database: a tool for genome-scale analysis of protein functions and evolution. *Nucleic Acids Research*, 28(1):33.

Tatusov, R., Koonin, E., and Lipman, D. (1997). A genomic perspective on protein families. *Science*, 278(5338):631.

Taylor, P. (1984). A fast homology program for aligning biological sequences. *Nucleic Acids Research*, 12(1Part2):447.

Thomas, J., Touchman, J., Blakesley, R., Bouffard, G., Beckstrom-Sternberg, S., Margulies, E., Blanchette, M., Siepel, A., Thomas, P., McDowell, J., et al. (2003a). Comparative analyses of multi-species sequences from targeted genomic regions. *Nature*, 424(6950):788–793.

Thomas, J. W., Touchman, J. W., Blakesley, R. W., Bouffard, G. G., Beckstrom-Sternberg, S. M., Margulies, E. H., Blanchette, M., Siepel, A. C., Thomas, P. J., Mcdowell, J. C., Maskeri, B., Hansen, N. F., Schwartz, M. S., Weber, R. J., Kent, W. J., Karolchik, D., Bruen, T. C., Bevan, R., Cutler, D. J., Schwartz, S., Elnitski, L., Idol, J. R., Prasad, A. B., Lee-Lin, S. Q., Maduro, V. V. B., Summers, T. J., Portnoy, M. E., Dietrich, N. L., Akhter, N., Ayele, K., Benjamin, B., Cariaga, K., Brinkley, C. P., Brooks, S. Y., Granite, S., Guan, X., Gupta, J., Haghighi, P., Ho, S. L., Huang, M. C., Karlins, E., Laric, P. L., Legaspi, R., Lim, M. J., Maduro, Q. L., Masiello, C. A., Mastrian, S. D., Mccloskey, J. C., Pearson, R., Stantripop, S., Tiongson, E. E., Tran, J. T., Tsurgeon, C., Vogt, J. L., Walker, M. A., Wetherby, K. D., Wiggins, L. S., Young, A. C., Zhang, L. H., Osoegawa, K., Zhu, B., Zhao, B., Shu, C. L., De Jong, P. J., Lawrence, C. E., Smit, A. F., Chakravarti, A., Haussler, D., Green, P., Miller, W., and Green, E. D. (2003b). Comparative analyses of multi-species sequences from targeted genomic regions. *Nature*, 424(6950):788–793.

Thompson, J., Gibson, T., Plewniak, F., Jeanmougin, F., and Higgins, D. (1997). The CLUSTAL_X windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. *Nucleic Acids Research*, 25(24):4876.

Thompson, J., Higgins, D., and Gibson, T. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673.

Thompson, J., Plewniak, F., and Poch, O. (1999a). A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682.

Thompson, J., Plewniak, F., and Poch, O. (1999b). BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87.

Tillier, E. and Collins, R. (2000). Genome rearrangement by replication-directed translocation. *Nature Genetics*, 26(2):195–197.

Treangen, T. and Messeguer, X. (2006). M-GCAT: interactively and efficiently constructing large-scale multiple genome comparison frameworks in closely related species. *BMC Bioinformatics*, 7(1):433.

Turnpenny, P. and Ellard, S. (2005). *Emery's elements of medical genetics*. Elsevier/Churchill Livingstone.

Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica*, 14(3):249–260.

Uricaru, R., Mancheron, A., and Rivals, E. (2010). Novel definition and algorithm for chaining fragments with proportional overlaps. In *Proceedings of Comparative Genomics, RECOMB 2010 International Workshop, RCG 2010, Ottawa, Canada, October 9-11, 2010*, Lecture Notes in Computer Science, pages 161–172.

Uricaru, R., Michotey, C., Noé, L., Chiapello, H., and Rivals, E. (2009). Improved sensitivity and reliability of anchor based genome alignment. In *Proceedings of the 10th Open Days in Biology, Computer Science and Mathematics (JOBIM)*, pages 31–36.

Van Walle, I., Lasters, I., and Wyns, L. (2004). Align-m–a new algorithm for multiple alignment of highly divergent sequences. *Bioinformatics*, 20(9):1428.

Varré, J., Delahaye, J., and Rivals, E. (1999). Transformation distances: a family of dissimilarity measures based on movements of segments. *Bioinformatics*, 15(3):194.

Venter, J., Adams, M., Myers, E., Li, P., Mural, R., Sutton, G., Smith, H., Yandell, M., Evans, C., Holt, R., et al. (2001). The sequence of the human genome. *Science*, 291(5507):1304.

Vishnoi, A., Roy, R., and Bhattacharya, A. (2007). Comparative analysis of bacterial genomes: identification of divergent regions in mycobacterial strains using an anchor-based approach. *Nucleic Acids Research*, 35(11):3654.

Vo, K.-P. (1986). More <curses> : the <screen> library. Technical report, AT&T Bell Laboratories.

Wallace, I., Blackshields, G., and Higgins, D. (2005). Multiple sequence alignments. *Current Opinion in Structural Biology*, 15(3):261–266.

Washietl, S., Hofacker, I. L., and Stadler, P. F. (2005). Fast and reliable prediction of noncoding rnas. *Proceedings of the National Academy of Sciences*, 102(7):2454–2459.

Wasserman, W., Palumbo, M., Thompson, W., Fickett, J., and Lawrence, C. (2000). Human-mouse genome comparisons to locate regulatory sites. *Nature Genetics*, 26(2):225–228.

Weiner, P. (2008). Linear pattern matching algorithms. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pages 1–11. IEEE.

Weinstock, G. (2007). ENCODE: more genomic empowerment. *Genome Research*, 17(6):667.

Wilbur, W. J. and Lipman, D. J. (1983). Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences of the United States of America*, 80(3):726–730.

Zhang, Y. and Waterman, M. (2003). An Eulerian path approach to global multiple alignment for DNA sequences. *Journal of Computational Biology*, 10(6):803–819.

Zhang, Z., Raghavachari, B., Hardison, R., and Miller, W. (1994). Chaining multiple-alignment blocks. *Journal of Computational Biology*, 1(3):217–226.