ASGART-fast & efficient de novo mapping of segmental duplications at the genome scale

Franklin Delehelle

▶ To cite this version:

Franklin Delehelle. ASGART-fast & efficient de novo mapping of segmental duplications at the genome scale. Quantitative Methods [q-bio.QM]. INSA de Toulouse, 2019. English. NNT : 2019ISAT0020 . tel-02497362

HAL Id: tel-02497362
https://theses.hal.science/tel-02497362

Submitted on 3 Mar 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Institut National des Sciences Appliquées de Toulouse

---

Présentée et soutenue par

Franklin DELEHELLE

Le 6 juin 2019

ASGART - Cartographie de novo des duplications segmentaires à l'échelle génomique

---

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Mathématiques et Applications**

Unité de recherche :
IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Hervé LUGA et Patricia BALARESQUE

Jury

Mme Macha NIKOLSKI, Rapporteure
M. Dominique LAVENIER, Rapporteur
Mme Sèverine BÉRARD, Examinatrice
Mme Émilie LECOMPTE, Examinatrice
M. Hervé LUGA, Directeur de thèse
Mme Patricia BALARESQUE, Co-directrice de thèse

Contents

1 Hominoids, Evolution & Segmental Duplications **15**
- 1.1 Variation Creation and Repeated Sequences . . . 15
- 1.2 Segmental Duplications . . . 19
  - 1.2.1 Definition . . . 19
  - 1.2.2 Dynamic & Consequences . . . 19

2 Segmental Duplications Mapping **25**
- 2.1 DNA Sequencing . . . 26
  - 2.1.1 Current State . . . 26
  - 2.1.2 Toward Third Generation Sequencing . . . 28
- 2.2 Formalizing Duplications Detection . . . 30
- 2.3 Algorithmic Components . . . 32
  - 2.3.1 Distances . . . 32
  - 2.3.2 Seed-And-Extend . . . 35
  - 2.3.3 Data Structures . . . 35
- 2.4 Existing Tools . . . 39
  - 2.4.1 Global Alignment Analysis . . . 40
  - 2.4.2 Database-Based Tools . . . 40
  - 2.4.3 NGS Data Processing . . . 41
  - 2.4.4 Short Repeat Searchers . . . 41
  - 2.4.5 Long Duplications Searchers . . . 43
- 2.5 Objectives . . . 46
  - 2.5.1 Computational . . . 46
  - 2.5.2 Biological . . . 47

3 ASGART **49**
- 3.1 Introduction . . . 49
- 3.2 Algorithm . . . 50
  - 3.2.1 Definitions . . . 50
  - 3.2.2 Pre-Processing . . . 51
  - 3.2.3 Clustering and Gathering . . . 52

3.2.4 Complexity 56
3.3 Implementation 57
3.3.1 Constraints 58
3.3.2 Technical Choices 61
3.4 User Interaction 64
3.4.1 Interface 64
3.4.2 Results Exploitation 66

4 Results & Discussion **71**
4.1 Benchmarking 71
4.1.1 Artificial DNA 71
4.1.2 Natural DNA 72
4.1.3 Performances Benchmarking 79
4.1.4 Parallel Scaling 81
4.2 Whole Genomes Benchmarking 85
4.3 Application 90
4.3.1 Duplications in Sex Chromosomes Evolution 90
4.3.2 Ongoing Work: Fertility Genes & SDs 95

5 Perspectives **97**
5.1 Computer Sciences 97
5.1.1 Algorithmic Improvements 97
5.1.2 GPGPU 98
5.1.3 UX, UI & Post-Processing 98
5.2 Biology 102
5.2.1 Hot Zones Demarcation 102
5.2.2 SDs Dynamics 103

Glossary **105**

Index **112**

Contexte disciplinaire des travaux

Cette thèse s'inscrit à l'interface de deux disciplines et instituts : l'informatique (INS2I) et la biologie (INEE). Elle est née de la rencontre entre des informaticiens intéressés par la biologie – et le traitement de grands volumes de données génomiques en temps contraint – et d'une biologiste intéressée par l'évolution des séquences complexes, *i.e.* répétées/dupliquées. L'explosion des technologies de séquençage haut-débit de troisième génération (type *linked* et *long-read*) va d'ici peu modifier notre rapport à la partie « complexe » du génome. Elles vont en effet offrir la possibilité d'analyser cette fraction d'ADN, restée jusque là inaccessible par le séquençage des *shorts-reads*.

Cette généralisation du séquençage des longs fragments et l'amélioration continue des techniques d'assemblage nous donneront l'accès à de nombreux génomes complets – régions complexes incluses – que nous pourrons comparer afin d'adresser ou ré-adresser des questions fondamentales liées à l'évolution des espèces. Dans ce contexte, ces travaux ont dû répondre à 3 défis : (i) l'appropriation du contexte biologique et sa maîtrise par un informaticien de formation, (ii) le développement d'un outil économe en ressource (temps de calcul et mémoire) permettant ainsi d'extraire les larges duplications d'un grand nombre de génomes totaux, (iii) l'application de cet outil à une question de biologie évolutive fondamentale.

Pour arriver à une co-construction de l'outil répondant à la fois aux contraintes computationnelles et aux spécifications de la problématique biologique, ces travaux se sont déroulés à temps partagé entre deux laboratoires toulousains, l'IRIT (Institut de Recherche en Informatique de Toulouse – CNRS UMR5505) et le laboratoire AMIS (Anthropologie Moléculaire et Imagerie de Synthèse – CNRS UMR5288).

Résumé

Le séquençage de nombreux génomes au cours de ces dernières décennies a permis de découvrir que la majorité d'entre eux recèle une large proportion de séquences dupliquées, et ce à diverses échelles. Parmi ces duplications, les DS (duplications segmentaires) sont d'un intérêt particulier de par l'influence qu'elles exercent dans les mécanismes de création de variation génétique. Ces zones, historiquement difficiles à séquencer, commencent à devenir plus facilement accessibles grâce à l'amélioration ou au développement des techniques idoines.

Pour faciliter l'exploitation *in silico* de ces nouveaux flux de données numériques, nous avons développé ASGART, un programme efficace, rapide, dédié à la détection précise des zones dupliquées dans des fragments d'ADN. Nous présentons ensuite quelques résultats préliminaires obtenus grâce à ASGART, en particulier dans le domaine de l'évolution des chromosomes sexuels.

Abstract

The large number of newly sequenced genomes during these last decades highlighted the large proportion of duplicated sequences – at many scales and scopes – they nearly all contain. Among these duplications, SDs (segmental duplications) are especially important, as they play a major role during the creation of genetic variation, both at the species and at the individual level. These areas, while historically difficult to sequence, are starting to be more easily accessible thanks to both the improvement of existing processes and the development of new ones.

To ease *in silico* exploration of these new datasets, we developed ASGART, a fast and efficient tool designed toward precise mapping of duplicated areas in DNA fragments. We also present a few preliminary results obtained thanks to ASGART, mostly in the field of the sex chromosomes evolutionary patterns.

Introduction

The idea of transmitting phenotypic traits from parents to their offspring through an inheritable vessel appeared as a side effect of the Theory of Evolution. DNA, its physical embodiment, was first isolated in 1869 by Friedrich Miescher. Its chemical structure, two complementary strings of nucleobases arranged in a double helix, was decrypted in 1953 by Watson, Crick, Franklin and Goslin; and the first genome sequenced – *i.e.* translated from its physical embodiment to a long string of letters representing its nucleobases – was the one of the Φ X174 bacteriophage, by Sanger and its team in 1977 [110].

This *première* was the starting point of a sequencing spree that persists to this day and saw dozen of increasingly complex and long genomes being sequenced, opening wide the gates to the fields of genetics and genomics. If a milestone was reached in 2001 with the first sequencing of an hybrid human genome [138], efforts are being pursued and sequencing technologies have continuously improved [48] to enhance the incoming data, both from a quantitative and a qualitative perspective.

Among the countless discoveries spawned by the studies of generated data, the high proportional quantity of duplicated fragments in virtually all studied genomes, from fishes[125, 126] to human[114, 12] through plants[88, 18, 19], was surprising. These duplications can be found at every scale, from a few nucleotides to whole genomes, and in counts ranging from a pair of duplicons to thousands of them. A classification for these duplications arose, depending on their duplicons length and count, their position, their biological role, and other characteristics.

Among these, our study will focus on *segmental duplications* (SDs). We found SDs to be an alluring subject, mainly due to the role they play in evolutionary matters[71]. This large class of duplications sits at a sweet spot regarding their length and mutation rate that makes them a major source of variation, both at the species and at the individual level. From a species perspective, it allows them to better adjust to their environment by the apparition of beneficial phenotypical variations.

But statistically, most of mutations tend to be nefarious to the individuals carrying them. Thus, SDs are of clinical interest from an individual perspective, as they are linked to several health concerns of various seriousness, including but not restricted to, reduced fertility or sterility[100], autism[73, 87, 108], Parkinson disease[102, 59], Charcot-Marie-Tooth syndrome[98], Alzheimer disease[103], and various cancers[26, 142] – this field being under-

standably mostly developed around the human species. SDs study is thus relevant both for fundamental, evolutionary fields, and applied, clinical ones.

Naturally, an SD can only be studied when having access both to the string of nucleotides its duplicons are made of, and to the location of these duplicons in the concerned genome. The former to determine their composition and the kind of genetic features they may encompass, the latter to understand their dynamic among the genome. Two broad categories of methods, with their respective pros and cons, can be used to map duplications embedded in a DNA fragment.

First, the *in vitro* methods act directly on actual genetic material. They, by nature, require a lab and the afferent logistics, as well as large quantity of high quality generic material, making them difficult to use. Second, *in silico* methods use computers to work on sequenced DNA fragments – the numerical representations of the physical molecule. Thus, they are twofold in their mode of operation: first, the concerned DNA must be sequenced by an hybrid hardware/software platform; then the resulting numerical dataset is analyzed by a software tool to gather the desired information. In contrast to *in vitro* solutions, *in silico* ones offer several advantages. They are cheaper, they do not require sampling apparatus and high quality biological material to extract, purify and store DNA (or, more precisely, they only need this step to be performed once before the sequencing process), are generally less time-consuming, and, last but not least, they are comparatively easily and cheaply reproducible.

Now that sequencing technologies are improving and starting to offer cheap, fast, high-precision coverage of sequenced genomes[48, 66, 77], the development of software tools especially dedicated to the *de novo* mapping of duplications is the next logical step. If some tools, such as WGAC [13], WSSD [12], or Vmatch [135], are aimed toward SDs mapping, they are typically unwieldy on large datasets, either due to the complexity of their pipelines and to the requirement for *in vitro* ressources, or to their intrinsic complexity.

To overcome this situation, we developed ASGART; a precise, fast, and efficient programs set dedicated to the discovery of medium- to large-scale duplications up to the multi-genome scale. ASGART is designed to make full use of the available hardware resources thanks to its parallel algorithm, that can take advantage of both multi-CPUs and multi-machines setups. It also features a simple command-line interface, as well as a web application, for an easy interaction. Finally, it makes use of standard data formats, both for inputs and outputs, to ensure a satisfying interoperability with the existing applications.

In this document, we will first present the biological, evolutionary background relevant to SDs research, and detail the roles they play according to the current state of the art. We will then focus on ASGART inner working after a review of the currently available bioinformatics tools in the domain. From there, following the presentation of several benchmarking studies, we will detail some preliminary results concerning SDs from an evolutionary perspective – obtained mostly thanks to ASGART – then conclude on some future development leads.

We would like to inform the reader that both a glossary and an index are available at the end of this document.

Chapter 1

Hominoids, Evolution & Segmental Duplications

1.1 Variation Creation and Repeated Sequences

Thanks to the recent progresses in sequencing technologies, genomes of individuals of every member of the hominoids family[1] have been sequenced at least to a draft level of quality. These new data provide an overview of an unprecedented chronological depth on the history of the human genome, as well as on the mechanisms driving variation and speciation concerning the hominoids.

A first observation has been the confirmation of the consistent presence in large proportions of repeated sequences of various scales and characteristics, akin to those that have been found in nearly all sequenced eukaryotes genomes to date[2][19, 125, 12, 126, 18, 114, 88]. These peculiar sequences families – be them low-complexity and high-repeat counts, or high-complexity and low-repeat counts – appear to be both disproportionally affecting and affected by recombination hotspots and other breakpoints.

Another result stemming from the comparison of these genomes shed light on the main ways of differenciation between these species. First, from a genetic perspective, a large majority of the coding sequences is nearly exactly conserved between these species. Thus, most of the difference is resulting from alteration not of the coding sequences of the genes themselves, but rather of their expression and regulation networks. Divergences in intronic and intergenic sequences thus explain most of the differences between hominoids. Second, at a smaller scale and from a genomic point of view, single nucleotide variations are utterly rare[3]. Most of the differenciation between once similar sequences is accounted for by larger scale events, mostly indels or rearrangements.

[1] Namely, the human, the chimpanzee, the gorilla and the orangutan.

[2] According to previous empirical observations, the actual quantity of such sequences is probably underestimated due to the challenges of sequencing and assembling duplicated sequences.

[3] An explanation for that could be the often deleterious consequences of such events.

The main mechanism at work behind the genesis of these events is recombination, which can be found in two major forms: either *allelic* or *non-allelic*. The former one generally results in symmetric (or nearly symmetric) outcomes, where the two sequences involved exchange an equivalent quantity of genetic material. But the latter will typically end up in asymmetric exchanges generating indels in the concerned sequences, with a strand gaining material and the other one losing an equivalent quantity. If allelic recombination is a common side-effect of homologous chromosomes pairing during the meiose, non-allelic recombination (or NAHR) is triggered under more restrictive conditions. It requires highly homologous but non-allelic sequences as a substrate as well as a failure of correction systems from the cell machinery to succesfully take place[1].

The dynamics that duplicated sequences typically assume as substrate during recombination events (Figure 1.1), combined to their presence in large quantities across the whole genome, make them major players in the aforementioned observations – both as subjects and catalysts – at scales varying from a few basepairs to several thousands at once. As subjects, their high rates of mutation per generation compared to the genome average testify of their volatility and thus their large sensitivity to variation creation processes, potentially affecting the surrounding genetic material as well. As catalysts, their repetitive nature combined with their layout make them an excellent substrate for incidents in recombination, leading to a dominant influence in *e.g.* NAHR or replication error events.

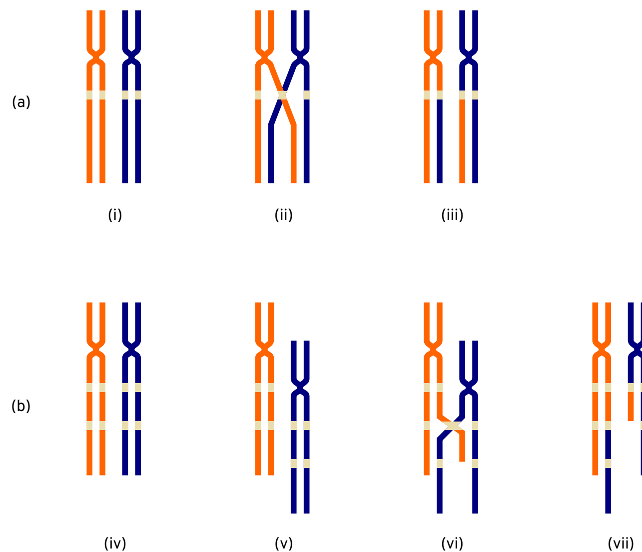
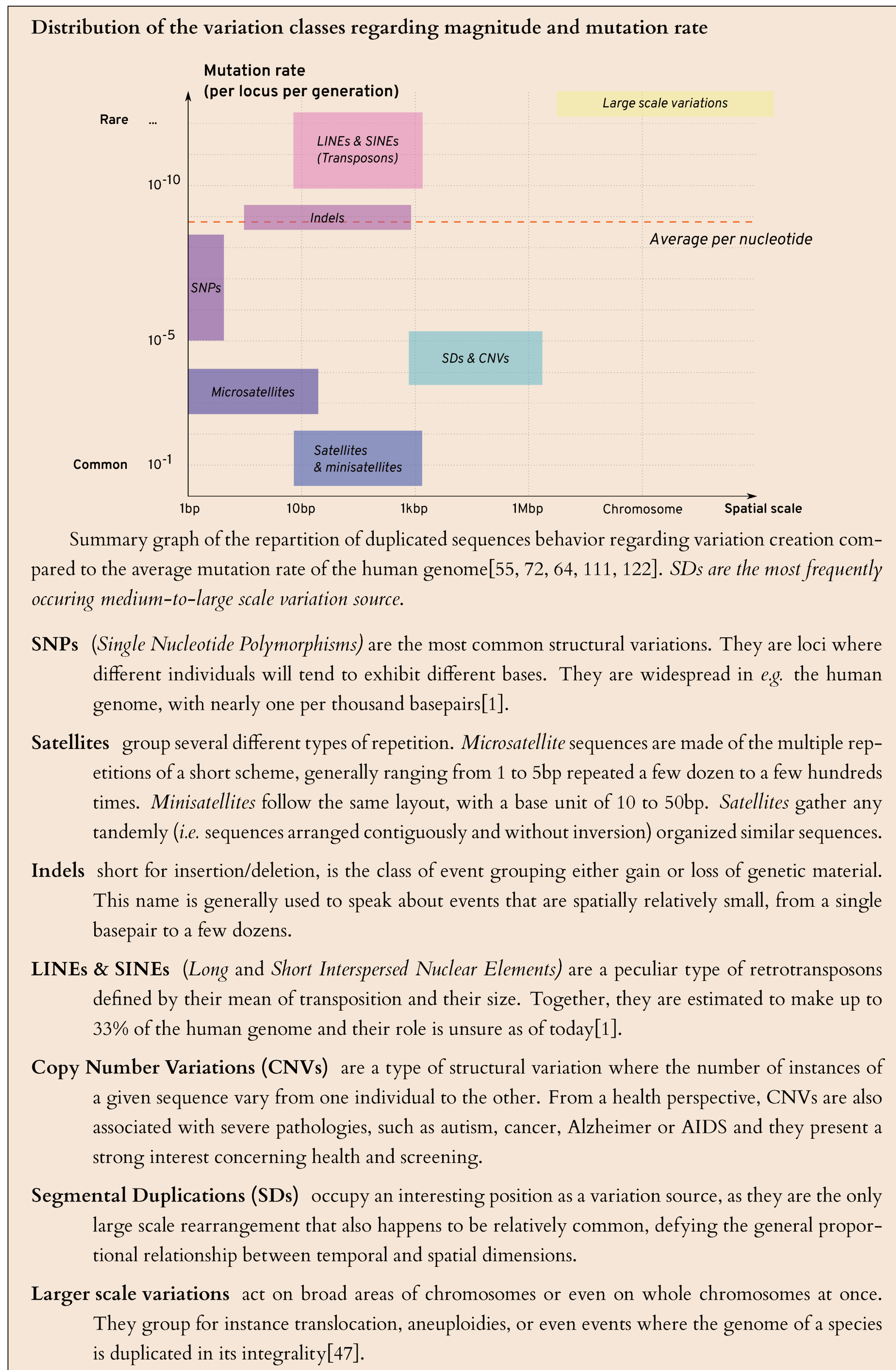


Figure 1.1: (a) homologous recombination resulting in a crossover event; (b) duplicated sequences are used as a substrate for **non-allelic** homologous recombination and lead to indel or large-scale rearrangement events.

Within the hominoids clade in general and for the human in particular, *Segmental Duplications* (SDs) – a class of repeated sequences – appear to play a capital role in genome plasticity and evolutionary dynamics. They exhibit characteristics, both in size, layout and content, that make them ideal to partake in events of significant phenotypic consequences.

We chose them as the focus of our study for their twofold consequences: first, as key elements in the evolutionary stories of these species; second, for the role they have concerning human health – both of these being the two faces of the same coin, *i.e.* their capacity to catalyze variation creation.

Distribution of the variation classes regarding magnitude and mutation rate



Summary graph of the repartition of duplicated sequences behavior regarding variation creation compared to the average mutation rate of the human genome[55, 72, 64, 111, 122]. *SDs are the most frequently occuring medium-to-large scale variation source.*

SNPs (*Single Nucleotide Polymorphisms)* are the most common structural variations. They are loci where different individuals will tend to exhibit different bases. They are widespread in *e.g.* the human genome, with nearly one per thousand basepairs[1].

Satellites group several different types of repetition. *Microsatellite* sequences are made of the multiple repetitions of a short scheme, generally ranging from 1 to 5bp repeated a few dozen to a few hundreds times. *Minisatellites* follow the same layout, with a base unit of 10 to 50bp. *Satellites* gather any tandemly (*i.e.* sequences arranged contiguously and without inversion) organized similar sequences.

Indels short for insertion/deletion, is the class of event grouping either gain or loss of genetic material. This name is generally used to speak about events that are spatially relatively small, from a single basepair to a few dozens.

LINEs & SINEs (*Long* and *Short Interspersed Nuclear Elements)* are a peculiar type of retrotransposons defined by their mean of transposition and their size. Together, they are estimated to make up to 33% of the human genome and their role is unsure as of today[1].

Copy Number Variations (CNVs) are a type of structural variation where the number of instances of a given sequence vary from one individual to the other. From a health perspective, CNVs are also associated with severe pathologies, such as autism, cancer, Alzheimer or AIDS and they present a strong interest concerning health and screening.

Segmental Duplications (SDs) occupy an interesting position as a variation source, as they are the only large scale rearrangement that also happens to be relatively common, defying the general proportional relationship between temporal and spatial dimensions.

Larger scale variations act on broad areas of chromosomes or even on whole chromosomes at once. They group for instance translocation, aneuploidies, or even events where the genome of a species is duplicated in its integrality[47].

1.2 Segmental Duplications

1.2.1 Definition

Segmental Duplications (SDs) are defined as several repeated units (not necessarily on a single chromosome), not transposons, at least 1Kbp long and presenting more than 90% of sequence identity between each repeated unit (or *duplicon*)[114, 10]. Known to be mostly present in human and African great apes genomes since their divergence from the Asian great apes, they are rather recent and, for the human, make up *ca.* 5% of the genome. Their origin is still unclear, and different models have been proposed to explain the origins[11, 85] of pericentromeric and subtelomeric SDs origins on the one hand, and, on the other hand, interstitial SDs.

By their very nature, SDs spread highly similar DNA fragments across the whole genome, in opposition to most other families of duplications, whose duplicons are organised in a tandem fashion. They represent a good substrate for NAHR, displaying a very high tendency to generate polymorphic inversions, insertions, and deletions – up to a ten-fold increase compared to the average[114, 41, 35, 10]. Given the scale of the variations they may catalyze, they play a role not only in structural variation, at the scale of the individual, but also, more generally, on the evolution of the species as a whole – a well-known example being the acquisition of trichromatic vision by the primates[32].

1.2.2 Dynamic & Consequences

Two peculiar cases in which SDs play a major role have been discussed in the literature, pointing to the potential further implications of this kind of duplicated sequences in peculiar dynamics: (i) the specific distribution of SDs in hominoids, and (ii) their dynamics in sex chromosomes.

Hominoids SDs

A first surprise that arose from the study of SDs was the considerable divergence between hominoids SDs and other mammals SDs.

Spatial Distribution First, hominoid-specific SDs differ from other SDs by their layout[85]. From studies on the most well-sequenced non-hominoid mammal genomes (namely the dog, the cow and the mouse), most of the SDs are found arranged in large tandem array zones, mostly gathered in subtelomeric and pericentromeric areas of their chromosomes.

In addition to the class of pericentromeric and subtelomeric SDs that they share with other mammals, they also feature so-called *duplication blocks* [4]. These blocks, instead of the simpler tandem arrays layout of pericentromeric and subtelomeric SDs, exhibit an intricate

[4] *Ca.* 400 of them are currently identified in the human genome.

structure. It seems to be the result of several rounds of duplications of the ancestral sequences together with the addition and embedding of DNA fragments of other origins[5]. The end result is a complex pattern of Russian nesting dolls duplicated duplications blocks specific to hominoids[65].

Like tandem SDs, these blocks tend to be mainly gathered in subtelomeric[6] and pericentromeric[7] regions, although a large amount of them (*ca.* 30%) is located on other parts of the chromosomes, distributed without an apparent pattern across the genome[10]. Interestingly, pericentromeric and subtelomeric blocks seem to have evolved differently than the other ones[63].

Content The hominoid-specific blocks of SDs also differ regarding their content. Whereas the SDs shared by all mammals generally contain mostly tandem-like structures of relatively simple, non coding sequences, these hominoid-specific blocks exhibit a far greater variety of content. They include the original duplicated blocks as well as more recent sequences, and can generally be found in both direct and reversed and/or complemented forms. Moreover, they are often (especially the intrachromosomal ones) enriched in non-expressed coding DNA and pseudogenes, usually in the form of disabled or degenerated intron and exon sequences. Hominoids-specific SDs contains more genetic material that has – or had once – coding properties, compared to the ones shared by all mammals, that are typically tandem arrays devoid of coding material [85].

These two axis of differenciation suggests two distinct evolutionary patterns for hominoid SDs: first an older one, shared with the other mammals for the pericentromeric and subtelomeric tandem-like SDs. Then a second one, specific to the hominoids, for their intrachromosomal duplications blocks. SDs falling within this second category are enriched in coding, or once coding, segments, and offer an improved genome plasticity to the concerned species[99, 51, 92]. They play a marked role in variation creation[10, 85], resulting in a fundamental shift in the dynamic of this kind of SDs compared to the other ones.

Sex Chromosomes Evolution: Acquisition of Chromosome-Specific Structures

In addition to the previously detailed particularities, primates exhibit an other specificity concerning their SDs. Publication of detailed maps of sex chromosomes of the human (both X[101] and Y [118]), the chimpanzee [56], and the macaca [57] have highlighted unusually large quantities of SDs on these chromosomes compared to the other ones in their genomes. Due to the role they play in sex determination, sex chromosomes are highly-valuable targets for the study of the explosion of SDs content in primates lineage.

Primates sex chromosomes have spawned from an ancestral pair of autosomes, when the emergence of a male-favorable allele (SRY) inhibited local recombination for this pair of

[5]*E.g.* retrotransposons or *Alu* sequences.
[6]For the human, *ca.* 40% of these blocks are situated in subtelomeric areas.
[7]Still for the human, *ca.* 30% of these blocks find themselves in pericentromeric areas.

chromosomes and sparked the XY sex-determination system in mammals. Through several large-scale inversions from the short to the long arm, this inhibition progressed in five waves along the two chromosomes, resulting in their current state: two distinct chromosomes with distinct evolutive histories and pressures, recombining only on their extremities (the PARs) and otherwise evolving mostly independently [76].

They still share a set of housekeeping genes stemming from the proto-sex autosome that require a stoechiometric dosage, forcing a presence on both of them. But otherwise, they now display different gene content, with the Y chromosome only keeping a stable set of 34 genes [7] out of the 640 it once shared with the X [58].

The Y Chromosome: Both Conservative and Dynamic? As an haploid DNA fragment exclusively transmitted from father to son through male lineages, the Y chromosome is a unique feature in the human genome. Besides this peculiar transmission system, its architecture is also remarkable, as (i) 15% of the chromosome still displays a very high homology with the X chromosome (Figure 1.2, *pseudo-autosomal* & *X-transposed*), and (ii) large SDs account for 35% of its sequence (Figure 1.2, *Ampliconic*). Roughly half (in base pairs count) of these SDs constitute eight large palindromes (Figure 1.3, Figure 1.2), P1-8. Not only are these palindromic SDs very long [8], they also feature extreme identity rates between their arms, from 99.94% to 99.997%.

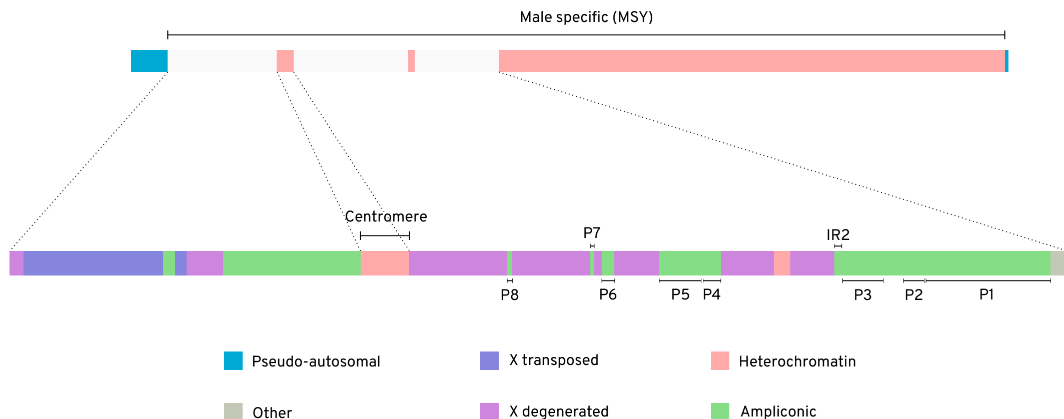


Figure 1.2: The composition of the human Y chromosome[118].

As a whole, the SDs of the Y chromosome contains a large numbers of genes (60 of the 78 genes known on the chromosome) involved in male fertility, many of them being multi-copies located on the duplicons of the SDs.

But despite duplicated sequences being a supposedly very favorable substrate for a high genetic activity through recombination [14], the human Y chromosome seems to display a highly preserved base structure [127], hinting towards a strong selective pressure to conserve this template.

[8] The arms of the longest palindrome, P1, are *ca.* 1.45Mbp long.

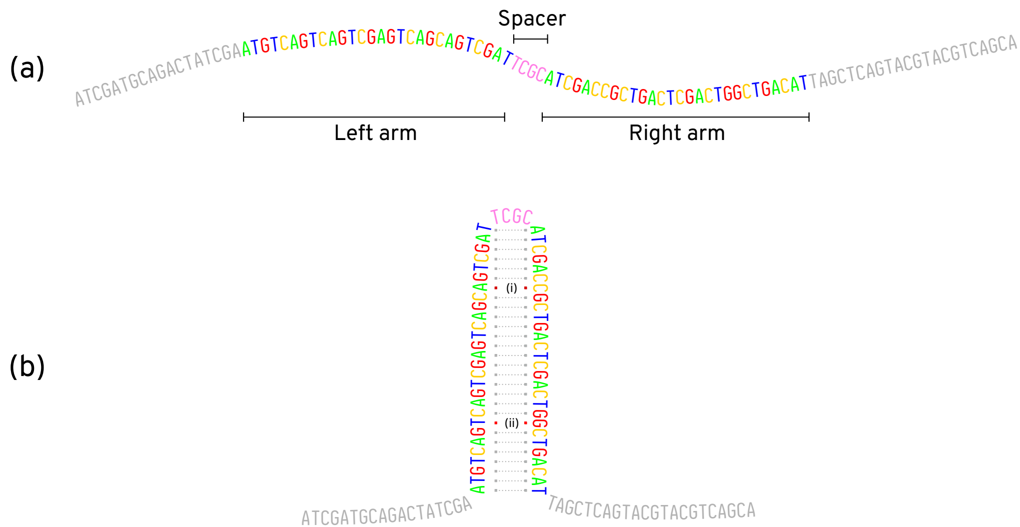


Figure 1.3: A palindromic sequence (a): the two arms, minus two mismatches ((i) A-C instead of A-T and (ii) T-G instead of T-A) are the reverse complement of each other, and thus can recombine together (b) if the chromatin were to fold in a favorable position. They are separated by a *spacer*, pictured in pink.

Although the MSY is not subject to traditional homologous recombination that would help in preserving its integrity, it seems to be the seat of frequent non-allelic homologous recombination events[104]. As recombination can happen on non-allelic sequences displaying a high identity rate, the large SDs of the MSY and its fragments homologous to the X are both good targets for recombination to take place; respectively endogenously between SDs duplicons, or exogenously with the X. Coherent with this background, high gene conversion rates have been observed on the human Y chromosome, with some of these events covering up to 10kbp at once with an occurrence rate of $2.9\text{-}8.4 \times 10^{-4}$ events per bp per generation [52] – to be compared with the conversion tract lengths of a few hundreds bp observed until then in meiotic division [62].

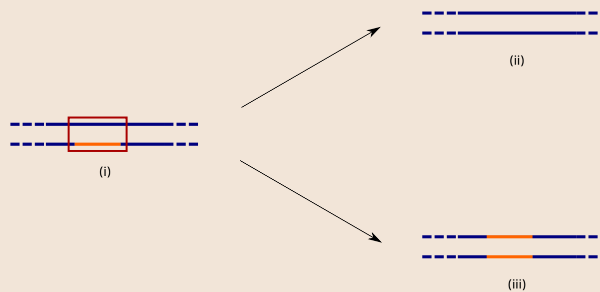
Taken together, these multiple peculiarities of the human Y chromosome hint towards a chromosome whose lack of an homologue to recombine with is compensated by an increase in activity of both NAHR and gene conversion – both endogenously within the SDs and exogenously with the X. Combined with a seemingly strong selective pressure, it results in a dynamic chromosome having exploited SDs to maintain the integrity of the important genes.

Now, the question is to determine whether, given their phylogenetic proximity and the fact their genomes share a quick enrichment in SDs, other primates Y chromosomes follow a similar scheme. It is already known that macacas and chimpanzees Y chromosome exhibit palindromic structures akin to the human ones, but further studies on these species are hampered by lacks in sequencing and duplications mapping technologies.

Gene Conversion

Recombination typically leads to formation of *heteroduplexes*; *i.e.* parts of a chromosome where the sister chromatides exhibit non-complementary DNA sequences. The cellular repair mechanisms triggered by these mismatches may result in *gene conversion*, another DNA shuffling mechanism[1].

The mismatching loci in the heteroduplexes will be excised from one of the sister chromatide, the other one being used as a template to synthesize DNA to fill the gap resulting from the excision. The choice of the strand being excised is seemingly random (although gene conversion tends to favor increasing the GC content of the genome[33]), so it may either fixate the external material acquired from the homologous chromosome during recombination or actually repair the chromosome to its original state. Depending on the conversion direction, the event will find itself either fixated in the gametes, or just cancelled. Gene conversion results in a non-reciprocal genetic information exchange from one of the chromosome to the other, on lengths typically ranging from 10 to 10,000bp. It is a highly active process in recombination hotspots or among paralogous sequences sharing a high identity rate[121].



Example of the gene conversion process. (i) This chromosome features an heteroduplex region, highlighted by the red frame. In the first scenario (ii), the repair mechanism used the first sister chromatide featuring the original state of the region as model for repair; thus, the variation introduced is lost. In the second scenario (iii), the new foreign fragment is used as a template, and the gene conversion occurs as the new version is fixated on the chromosome pair.

Chapter 2

Segmental Duplications Mapping

Given the important role played by SDs in recombination dynamics, it is no surprise that they represent a hot topic; both from an applied perspective (mainly health, *e.g.* personalized medicine or research) and from a more fundamental one (*e.g.* phylogenetic markers, recombination prediction, modeling of gene evolution, etc.). But in order to be able to study SDs, they must be mapped onto the studied genome. Once their position is known, their DNA sequences can be extracted and they can be compared with other genomic components, such as genes position, regulatory network, recombination hotspots, and so forth. But problems that may be, although tedious, easy to solve on small datasets, may become exponentially more complex on longer one. Detecting highly – but not exactly – similar fragments of DNA in or across genomes, whose typical size are in the magnitude of several billions of bases pairs, is actually a real challenge.

The traditional approaches used to be *in vitro* ones, as they can work directly on the DNA molecules without the need for the complex phase of sequencing, a process that used to be expensive, error prone, and time-consuming – and still is, although in smaller proportions. However, *in vitro* methods suffer from several practical drawbacks:

- laboratory, personnel, and the logistical tail requires are maintenance-heavy and costly;
- an access to the source DNA material is needed, which is not necessarily easy;
- their reproducibility is rather low, due to their analog nature;
- parallelization and automatization opportunities are close to zero;
- they offer a crude resolution, typically up to a few thousands base pairs.

Therefore, the use of computer programs to process genetic data appeared immediately after the first sequencing processes were designed, as their digitization allowed for the development of automatized, parallel, unsupervised programs to operate on these data.

In silico methods regroup all processes able to find duplications based only on computer-based processing of sequenced DNA, therefore not including hybrid methods, like computer-driven post-processing of FISH images. However, as these methods work on sequenced, digitalized DNA, they of course depend on the quality and precision of the sequencing process. Unfortunately, there are no ideal sequencing pipelines today able to, being given any DNA macromolecule, output an exact transcription of its nucleotide sequence – despite steady progress in the field. A description of common sequencing methods and their compromises will help understand the challenges and peculiarities of working on sequenced DNA, that are reflected in many bioinformatics subfields.

2.1 DNA Sequencing

2.1.1 Current State

DNA sequencing is the field of research dedicated to develop and improve means of translation from DNA macromolecules to digital sequences of letters matching the base pairs of the DNA. As of today, none of them can actually process long DNA fragments at once, and most popular techniques of the last decade focus on sequencing shorter overlapping *reads* that are then linked together to form an *assembly* of the longer sequenced fragment (Figure 2.1). These *shotgun sequencing*[1] methods are mainly characterized by their *reads length*. Depending on their technical characteristics, sequencing methods can be used either as *de novo* method, to sequence new genomes, or to sequence genomes of individuals of species for which a reference genome is already available and on which the reads can be mapped.

DNA sequencing was pioneered by Frederick Sanger and his team in the late seventies and their method[109] was the foundation for the first generation of sequencing methods. This generation was architectured around the partial sequencing of multiple amplified fragments of the target sequence. Each one would be amplified many times and see its nucleotides determined individually by a partial polymerization of the denatured molecule with marked terminators nucleobases. Once all of these fragments had been independently sequenced, they would be chained together thanks to their overhanging parts to form the final result.

Many technical improvements were gradually introduced, and eventually allowed for the first sequencing of an hybrid human genome assembled from multiple donors in 2001. This method allows for precise sequencing of reads up to a few hundreds bases long, after which errors rate increase too fast for accurate sequencing. However, it is slow, costly, and neither fast nor easily parallelizable. It is still in use today in the same conceptual shape, mainly for *de novo* sequencing thanks to its relatively long average read length.

The ever increasing need for faster and cheaper sequencing means gave birth to what is commonly referred as the *New Generation Sequencing* (NGS) technologies, or *high throughput sequencing*, due to their extremely fast working speed when compared to older processes.

[1] Comparing the spread of the countless short reads to the dispersion pattern of a shotgun.

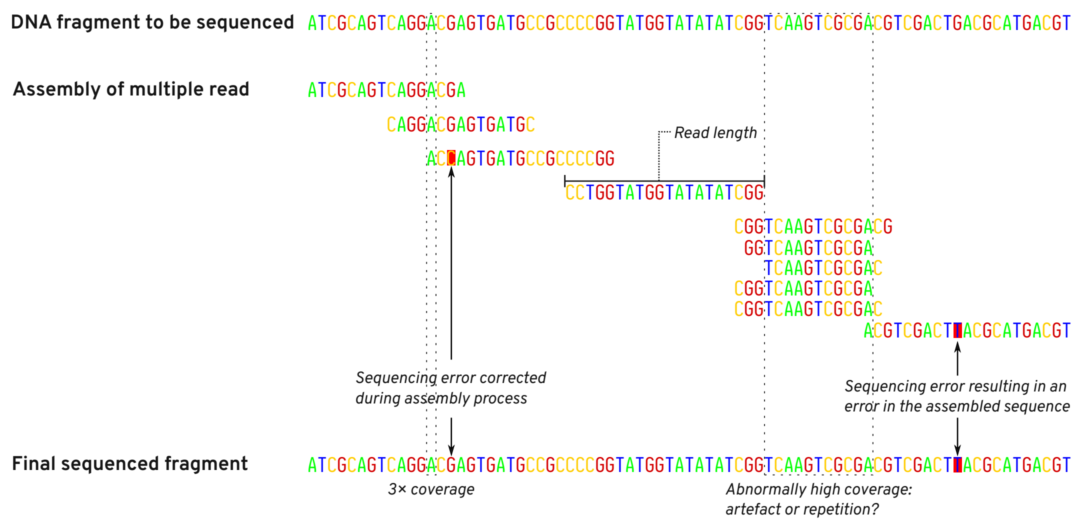


Figure 2.1: Overview of the sequencing process. A DNA fragment is, due to technical limitations, not sequenced at once; but rather sampled by multiple overlapping short reads, that are then stitched back together in an assembly thanks to their overhang. As the generation of these reads is never flawless, they all may contain errors. These errors can be fixed *e.g.* by choosing the consensus when coverage is sufficient. But if a sequencing error happens in a poorly-covered region, there is often no way to detect and fix it. Assembly algorithms have difficulty telling apart dulications from artefacts in coverage due to the short read length. This figure is but a schematic representation for clarity sake; the actual process entails hundreds of thousands or millions of short reads spanning between dozens and hundreds nucleobases (depending on the method), and the coverage ranges typically in the dozen.

These methods take advantage of the already sequenced, assembled genomes to focus on speed, parallelization, and cost-reduction. Instead of aiming for the sequencing of a few relatively long fragments and assembling the resulting genome from there, they rather focus on producing really quickly millions of short reads (typically up to 100bp) and produce the final result either by aligning them on a known *reference genome* (which is more than often obtained through Sanger sequencing), or by linking them together thanks to their overlapping parts.

Therefore, they are mostly useful to study structural variations or genomes that are close enough to an already sequenced one. For instance, a team working on the ancestral form of a species could sequence ancient DNA and assemble the resulting reads thanks to the current-era species genome, as they are expected to be very close. The most common method is *Illumina*'s *sequencing by synthesis*[60, 48], although other methods offering different compromises between precision, cost, speed, throughput, and other characteristics exist, such as pyrosequencing[96], ion sequencing[61], and others[48].

2.1.2 Toward Third Generation Sequencing

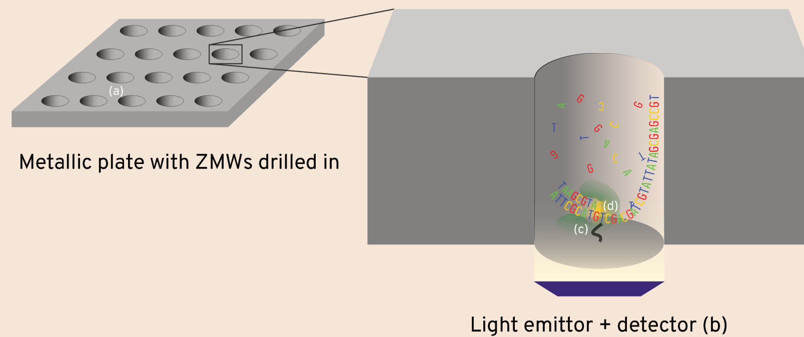
All NGS methods follow the same meta process, *i.e.* produce thousands or millions of tiny reads, that are then aligned to a reference genome. Although these methods marked a great leap forward in sequencing availability and affordability, they are handicapped by the short length of their reads. It often restrains their use to cases where a reference genome, either from the same species or from a very closely related organisms, is available to make sense of the millions of available reads that would otherwise be next to impossible to assemble.

Indeed, a difficulty arises from the limited length of the reads, that make them not adapted for studying or sequencing duplicated areas of the genome *de novo* . They are arduous to assemble correctly, as assembly programs will have a hard time determining if highly similar reads are actually different and part of different duplicons, or just random higher coverage of a single area. Therefore, the need for sequencing processes resulting in longer reads arose. The existence of such a process would greatly simplify and improve reference genome assemblies and drastically enhance current assemblies of highly duplicated areas of these genomes.

Even if there is currently no solution completely solving these problems, two promising approaches, while not flawless, offer enthusiasming perspectives. *Oxford Nanopore* and *SMRT*, additionally to offering reads lengths flirting with the dozen of thousands of base pairs, also work closely to the natural *in-vivo* speed with a single molecule of DNA, therefore bypassing the long and expensive phase of amplification of the previous methods: with these new tools, one just needs an isolated and purified molecule of DNA. Although it will still have to be broken down in multiple fragment if it exceeds the maximal read length, it will mark a tremendous improvement and simplification over 2nd generation methods.

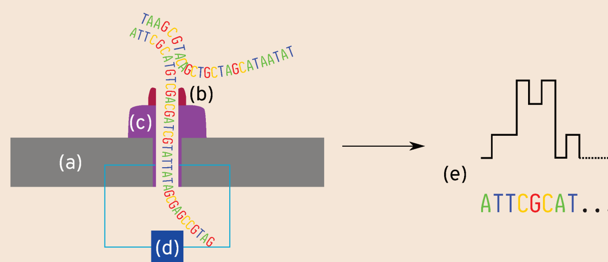
Third Generation Sequencing Methods

SMRT



Schematics overview of the SMRT sequencing method. (a) the plate with its ZMW wells; (b) the light emitter + detector electronics is located directly beneath the plate; (c) the polymerase is fastened to the bottom of the ZMW, in a solution of fluorescently-marked nucleotides; (d) every time it links a new nucleotide to the sequenced DNA fragment, the nucleotide is maintained long enough at the bottom of the well for the electronics to record its signal and differentiate it from the ambient noise. Many (typically several thousands) ZMW wells can be laid out on a single plate, and polymerization reaction is going roughly at the same speed than *in vivo* (*ca.* 50bp/s), resulting in a high throughput with read lengths going up to 10-50Kbp.

Nanopore



The DNA strand is denatured by an enzyme (b) and is threaded through the nanopore (c) in an electrically charged plate (a). The resulting variation in current is measured (d) then translated to a numerical format (e). The fast processing of each nanopore combined with the massive parallelization (*i.e.* number of pores working concurrently on a single plate) culminate in a very high real-time throughput for read lengths ranging up to 200Kbp.

Unfortunately, as of today, both SMRT and Nanopore exhibit average error rates going over 10%, and therefore need further refinements to compete with the less than 1% of *Illumina*[48].

But with the rapid development of the field, the technologies should hopefully soon be able to satisfy these precision constraints, and terabytes over terabytes of new genomic data will then flow in, necessitating always more software pipelines to process them adequately, and efficiently play the needle-in-the-haystack game that is exploratory genetics.

2.2 Formalizing Duplications Detection

As previously detailed, segmental duplications are strongly suspected to play a very active role in some species evolutionary story. But as mentioned earlier, they are only a single type of duplicated sequences among many others: although this study focus only on SDs, all duplications have a role to play; be it in evolutionary studies, genetics, health or forensics.

Thus, repeated sequences detection is an old domain within the frame of genome analyses. From a theoretical perspective, all repeats of a string S defined on an alphabet Σ can be formalized in the same way (Figure 2.2), as a set of n non-intersecting subsequences $\{s_i\}$ of length l_i, and separated by a set of $n-1$ distances d_i. To comply to the biological definition of an SD, each of these subsequences should also exhibit an identity rate with each other greater than a given threshold h, either directly, or once reversed and complemented to account for palindromic SDs. It is assumed that although a genome is typically physically split in chromosomes, it can be, once sequenced, considered as a connex, unified search space of a single large string. Chromosomes are then modeled as a set of indices pointing inside this large string.

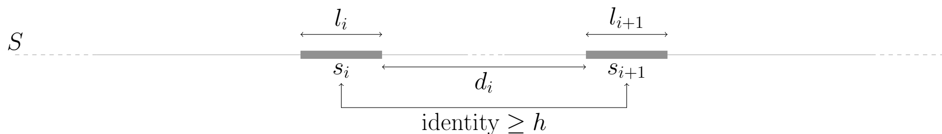


Figure 2.2: Schematization of two duplicons i and $i+1$ of a duplications family, according to our proposed modelization of duplications.

The many different kinds of repeats – as well as their sub-variants – can all be described by this model. Whole chromosomes duplications, for instance, are modeled with $n = 2$, $h = 100\%$, $d = 0$ (assuming the duplicated chromosomes are put next to each other in the sequenced genome), $l_1 = l_2$, and Σ being the standard nucleobases alphabet. On the other side of the scale spectrum, a microsatellites family would be described, using the same S and Σ, with a count of repetitions n ranging from the dozens to the hundreds, a set $\{l_i\}$ of a handful of basepairs, a set of distances $\{d_i\}$ scoring from zero to one, and a distance threshold h between the duplicons close or equal to zero.

Table 2.1: The range of values accessible to the parameters of the model we present.

Nomenclature	Characteristic	Experimemtal domain
Σ	Alphabet	DNA, RNA, proteins
n	Duplicons count	2 – 1000+
l_i	Duplicons length (char.)	2 – 1,000,000+
h_i	identity rate (%)	100 – $\sim$70
d_i	Distance (bp)	0 – 1,000,000+

From these examples, one can see the first challenges any implementation will soon meet: the multiple classes of repetitions cover a large range for each of these model parameters, as shown in Table 2.1. A duplication can be made from two to thousands of duplicons, either directly following each other or wildly spread across a genome, either identical or degenerated, from wildly varying sizes, and so forth. Moreover, the search space itself (*i.e.* S and Σ) can take many forms, from small prokaryotes genomes of a few dozens thousands basepairs to large eukaryotes ones, ranging in several billions basepairs; or even on small proteins encoded in the protein alphabet.

If it would be theoretically possible to design a "master" algorithm able to encompass all of the use cases and detect any kind of duplications in any source material, it would however reach algorithmic complexities levels that would make it close to useless for any real-world application. Thankfully, the many distinct classes of duplications may be leveraged to form a partition of the solution space, allowing for the development of algorithms and programs specialized in one or several of these partitions only, and thus able to make use of optimization methods that would conflict with other partitions constraints – should they all be satisfied at once.

Let us now focus on the problem space relevant to our project, namely the search of duplications in mammals and prokaryotes genomes; *i.e.* strings built from the nucleobases alphabet, typically ranging in the billions of basepairs. We found two main computational classes (although each of them could be split further) of duplications in this search space. It should be noted that these classes were built around *computational* concerns, and although they form a partition of the solution space as biological classes do, they do not coincide with them.

The first class contains the duplications made, according to our model, of repetitive areas exhibiting large repeat numbers (from the dozens to the thousands), small lengths (from a few basepairs to a few dozens), small to nonexistant gaps between each other, and a very high identity rate, equal or close to 100%. In a biological perspective, such repeats will encompass microsatellites, minisatellites, satellites, poly-A tails, STRs, and other high-frequency, low amplitude repeat classes. Historically, these kinds of repeats were the first to be discovered and studied in the then-newly sequenced genomes, as their high frequency and identity rate made them easily discernible with a naked eye.

The second class contains longer, more disparate repeats. Following our model, they would be made of relatively low numbers (two to a few dozens) of large repeats (a few hundreds basepairs up to several hundreds thousands basepairs) separated by large areas of genetic material (from a few hundreds basepairs to several chromosomes) and display falling identity rates, down to 70% in the most degenerated cases, *e.g.* pseudogenes. Interestingly enough though, experimental observations showed that the discrepancies between duplicons accounting for these low identity rates tend to be grouped, be it in islands of clustered alterations or large insertions or deletions on some of the duplicons, the remaining parts exhibiting

far higher local identity rates. This peculiarity is paramount to the establishment and optimization of the later-detailed "seed-and-extend" strategy used by many duplications finder, including the one we present in this document. This class overlaps with several biological classes, including but not restricted to, LINEs, SINEs, SDs, gene families, and so forth. Due to the looseness of their criteria, their detection and mapping raise several computational challenges, as the large dimensions of the search space involved leads to a dramatic drop of performance when confronted with real-world datasets. However, SDs are a member of this class (Table 2.2), and so we must tackle the challenges it implies to pursue our study.

Table 2.2: Formalization of the SDs according to our model. Description of microsatellites according to the same model is presented for contrast.

Characteristic	**Microsatellites**	*SDs*
Alphabet	DNA	*DNA*
Duplicons count	5 – 50	*2 – 10+*
Duplicons length (bp)	1-10	*1000 – 1,000,000+*
Identity rate (%)	~100	*>90*
Distance (bp)	0	*1000s – 1,000,000+*

2.3 Algorithmic Components

Algorithms typically rely on the reuse of already existing building blocks to avoid duplicating efforts and focus on their own goals by leveraging prior work. In accordance with this principle, a number of fundamental theoretical concepts, algorithms and data structures are shared among the many duplications finders currently published.

2.3.1 Distances

In our modeling of duplicated sequences, we introduced the biological concept of identity rate between sequences, which is formalized thanks to the mathematical concept of distance. A distance is a function d defining a (generally scalar) distance between any pair of elements (x, y) of a given set, satisfying the following conditions:

- $d(x, y) \geq 0$;
- $d(x, y) = d(y, x)$;
- $d(x, z) \leq d(x, y) + d(y, z)$;
- $d(x, y) = 0 \Leftrightarrow x = y$.

The last point might be troubling, as two distinct duplicons formed of the exact same DNA string will have their distance equal to zero. It is, indeed, true that their representation as

strings are identical; however, the biological objects they represent are not. It should thus be noted than a zero distance, or a 100% identity rate, between two duplicons only means that they belong to the same equivalence class under the relation $d(x, y) = 0$ and that their string representations are equals, although the two duplicons mamay be different biological objects.

Hamming Distance

A first distance used in the study of repetition is the Hamming distance[53]. It is defined as the sum of differing characters in two strings defined on the same alphabet, and can obviously be used only on strings of identical lengths. The biological identity rate is computed from this distance by 100-complementing it and then weighing it by the length of the strings. Practically, it can only be used on repetitions where nearly every duplicon is expected to be of the same length than the others, this restriction limiting its use mainly to STRs and other microsatellites structures. In our context, *i.e.* the study of SDs, where the length of every duplicon is not expected to be identical to its brethren[2], this metric is not usable.

Levenshtein Distance

When measuring the distance between strings of variable length, the most common metric is the Levenshtein distance[105]. It is defined as the minimal number of single-characters alterations required to transform one of the string into the other. Akin to the Hamming distance, it is often weighed by the length of the shortest string and 100-complemented to obtain the identity rate. Its fundamental difference with the Hamming distance is that it takes into account all types of single-character edits, whereas Hamming's only accounts for substitution, *i.e.* changing a character into another one inside the considered alphabet (Table 2.3); all of its other characteristics are mere consequences of this feature.

Table 2.3: Comparison of the Hamming and Levenshtein distances over several sets of DNA strings. Mismatches contributing to the distance are marked in bold.

	ATTG ATTC	ATTAC AATTA	ATTA ATA
Hamming	ATT**G** ATT**C** 1	A**TTAC** A**ATTA** 4	Undef.
Levenshtein	ATT**G** ATT**C** 1	**-**ATTA**C** **A**ATTA**-** 2	AT**T**A AT**-**A 1

To do so, Levenshtein distance also reckons indels in addition to substitution. This seem-

[2]Mostly due to the aforementioned clusters of indels.

ingly slight difference makes it very polyvalent, but also far more complex to compute. It is generally not directly used for duplications detections out of obvious concerns regarding the complexity of its computations, but is generally the go-to tool to refine the results, after they have been found with a heuristic method.

Needleman-Wunsch Algorithm

The complexity of the computation of the Levenshtein distance is high, both in terms of time and memory consumption. The most used algorithm, the Needleman-Wunsch algorithm [94], as well as its local-alignment aimed *alter ego*, Smith-Waterman [140], are based on dynamic programming.

To compute the mathematically optimal alignment of two string S_1 and S_2 (example Table 2.4), respectively n_1 and n_2 bp long, a matrix $M \in \mathbb{R}^{n_1+1 \times n_2+1}$ is created. It is then recursively filled according to a scoring system accounting for matches, substitutions, and gaps opening and closing (Equation 2.1), with $S(x, y)$ being the score penalty for the substitution of nucleotide x to a nucleotide y, and d the score penalty for opening a gap. The first argument of the max function corresponds to the introduction of a (mis)match in the final alignment, the second argument to the introduction of an insertion in the first string, and the last argument to the introduction of an insertion in the second string – or, conversely, of a deletion in the first string.

The algorithm may be fitted more precisely to different tasks by playing on the scoring system, *e.g.* by using different penalties for different nucleotides mismatches or adapting the gap penalty to the problem at hand.

$$\begin{cases} M_{0j} = d \times j \\ M_{i0} = d \times i \\ M_{ij} = \max(M_{i-1,j-1} + S(S_{1i}, S_{2j}), M_{i,j-1} + d, M_{i-1,j} + d) \end{cases} \tag{2.1}$$

Table 2.4: Example of the Needleman-Wunsch algorithm being used to compute the optimal alignments of the sequences ACAA and ACTGA. The circled numbers mark the cells used to compute the final alignment, *i.e.* ACA-A/ACTGA.

		A	C	A	A
	0	-1	-2	-3	-4
A	-1	①	0	-1	-2
C	-2	0	②	1	0
T	-3	-1	1	①	0
G	-4	-2	0	⓪	0
A	-5	-3	-1	1	①

To find the optimal alignment(s) from this point, a path is carved through M from the bottom-right-most cell, moving one cell at a time, to the left, top, or top-left cell, depending on which one was used in Equation 2.1 to compute the current cell value. In cases of branching, the multiple nascent alignments can be independantly explored. Finally, the definitive aligned sequence(s) are built from this (or these) paths.

Both computational and space resulting complexities are $\mathcal{O}(n_1 \times n_2)$ for the first widely used algorithm, Needleman's and Wunsch's[94].

A later improvement, Hirschberg's algorithm[54], further enhanced by Myers and Miller[93], decreases memory consumption complexity to $\mathcal{O}(\min(n_1, n_2))$. But although its computational asymptotical complexity remains unchanged, it tends to, unfortunately, behave worse than its predecessor on real-world data[36].

2.3.2 Seed-And-Extend

Even with these improvements, looking for similar areas in or across whole assembled genomes by directly comparing their subsequences according to the Levenshtein distance is unfortunately not within the realm of possible with the available algorithms and hardware – and this situation is not expected to significantly improve[8].

Thus, heuristic algorithms are needed to solve the problem in acceptable computational and space complexities. Naturally, an heuristic algorithm is never as good as an exact one; but a peculiarity of the observed duplications in the already studied genomes across many organisms points to a clue that heuristic algorithms might obtain excellent results despite their imperfection.

As mentioned before, studying large duplications shows that mismatches and indels between their duplicons are not equally distributed among the length of the duplicons, but tend to be gathered in relatively large islands of indels, the major remaining parts of the duplicons being close to identical. Thus, a common solution adopted by many duplications finding programs is to look for closely similar substrings in the reference string thanks to a faster, heuristic algorithm. These restricted areas are then clustered and further processed with more precise, albeit costlier, methods offering a better resolution on these subsets of the solution space.

2.3.3 Data Structures

With its challenges stemming from similarity finding and strict or loose string matching, many subfields of bioinformatics (and, especially, duplications finding) intersects strongly with stringology. Besides the aforementioned distances, quite a few data structures stemming from there proved to be of great usefulness.

Suffix Trees

As previously mentioned, a common basis for loose duplications finding algorithms is the seed-and-extend strategy. The underlying concept is quite simple, and close to divide-and-conquer. It splits the problem in two sub-problems: the first one is to identify exactly matching substrings of the reference genome; the second one is to cluster or link together these identical substrings to detect islands in the source material exhibiting a circumstantially low distance between each others, working on a drastically reduced subset of the problem space. During the first phase, searching for identical regions requires fast algorithms using little memory, as it is performed on the full problem space – which might scale up, in the case relevant to our study, to the multi-genome dimension.

A commonly encountered structure filling these criteria is the *suffix tree*. A suffix tree[141] (an example is featured in Figure 2.3) is a tree encoding all the suffixes of a given reference string. It is built so that, starting from the number n found on one of the leafs, tracing the path back to the root while concatenating in reverse order the strings found along the edges will build the nth suffix of S, or $S[n...]$.

Many methods to build such a tree from a reference string have been developed [86, 5, 136, 68, 69], but they all result in a tree satisfying the following criteria:

- every edge is labeled with a substring of the original string;
- every leaf contains the index in the reference string of the string created when concatenating letters found on edges from the root to this leaf;
- every node has at least two children;
- there are exactly as many leafs as there are characters in the indexed string.

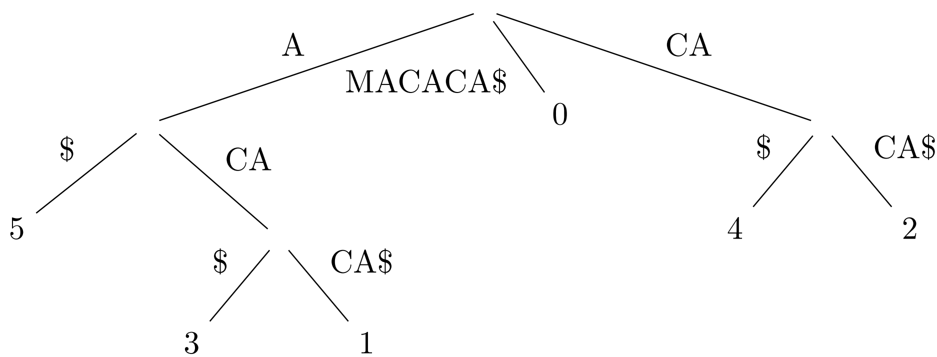


Figure 2.3: Simple suffix tree for the word "Macaca". A dollar sign has been concatenated to mark the end of the indexed string.

Once a query, or needle, string is established, searching for all its occurrences in the reference string is done with a simple tree traversal (Figure 2.4), whose time complexity is

$\mathcal{O}(n)$ regarding the length of the query string. If the construction of the tree itself might be costly, what matters is that its final spatial complexity is linear regarding the size of the reference sequence, and that its time of construction is small compared to the whole running time of the duplications finding program using it, conditions that are satisfied thanks to the most recent progresses in the matter[37, 116].

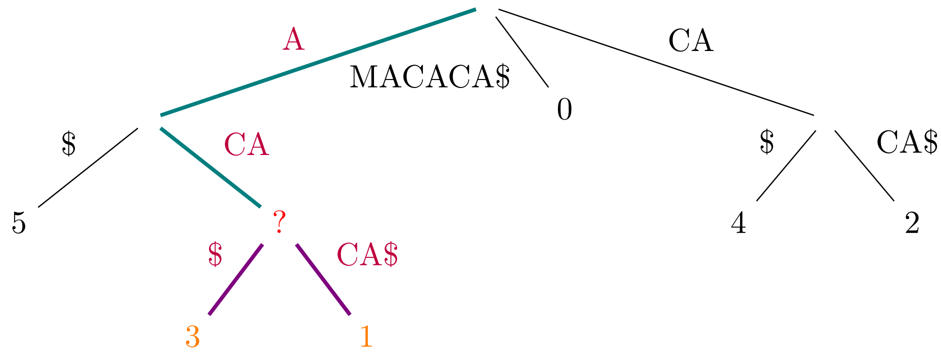


Figure 2.4: An example search of all the occurrences of “ACA” in the reference string “MACACA” using a suffix tree. Starting from the root, edges forming the query string “ACA” (teal) are followed, ending at the node marked with a red quotation mark. From there, the recursive exploration (in purple) of all children nodes to their end yields the position (in orange) and content (in crimson) of all substrings starting with “ACA” in the reference string, namely `ACA$` starting at position 3, and `ACACA$` starting at position 1 (the dollar sign is used as an end-of-string marker).

But however efficient they are, suffix trees suffer from some inherent flaws that make them costly, as well as suboptimal in our case. First, the tree requires at least as many nodes as there are nucleotides in the reference DNA string. And each node stores some data, such as *e.g.* the letter it refers to, pointers to its children, potentially to its father, and so forth. Therefore, this payload size is a linear coefficient multiplying by so much the memory consumption of the tree regarding the length of the reference string.

Moreover, a tree traversal, although an algorithmic very efficient operation, hides a major flaw concerning current most-used CPU architectures: random accesses in the large memory block(s) in which the tree is stored are very cache-unfriendly operations, preventing the program to use the CPU at the best of its capacities. The situation concerning these problems was improved by the development of the *suffix arrays* [84], a derivative of suffix trees. They are a very simple data structure, a flat array the size of the reference string, where each cell contains the ith prefix of the reference string taken in lexicographic order.

Suffix Arrays

Although they offer less opportunities of use than the suffix trees, they retain the main functionality required in our context, *i.e* a fast search for exactly matching substrings of a large reference string. A suffix array is made of the indices of all the suffixes of a string, but sorted

lexicographically (example Table 2.5).

Table 2.5: The suffix array for the word "Macaca". Typically, out of memory concerns, only the second column (*i.e.* the actual suffix array) is stored; the elements of the third one are built on the fly when needed.

Index	Suffix Array	Corresp. substring in reference
0	5	A
1	3	ACA
2	1	ACACA
3	4	CA
4	2	CACA
5	0	MACACA

In a perspective limited to exact substrings matching, they kill two birds in one stone. First, they only need exactly one word (typically a 32bits one, enough to index genomes up to ~4.29Gbp) per nucleotide in the indexed genomes. Second, they have a more cache-friendly behavior than the suffix trees, as they are more compact than the suffix trees and are typically linearly stored in memory. Thus, a larger part of them can fit in the same number of cache lines. However, this peculiarity would be close to useless if a search operation were to access random parts of the whole array. But a search operation on a suffix array is typically made through a binary search algorithm. The binary search algorithm (a type of dichotomic search) starts at the middle of a **sorted** array, and compares the element there to the target with a boolean function defined on the space of the array elements. Depending on the result, it will then reiterate the same operation either on the remaining top or bottom half of the array when the comparison is respectively "less than" or "greater than". Once the searched element has been found, adjacent elements are tested for equality and included in the result accordingly.

Binary search time complexity is logarithmic [40] and the algorithm is rather cache-friendly: the first queries hit far away parts of the array, but the geometrically decreasing size of the partitions allows a better use of the cache lines. Of course, if the binary search in the array is cache friendly itself, it implies no such thing for the element-to-query comparisons needed at each step of the search. But these comparisons being simple string-to-string comparisons on relatively small strings (typically ten to twenty characters), they can be accelerated thanks to the use of SIMD operations, found in instructions sets such as SSE or AVX for x86-64 or NEON for ARM, available on any reasonably recent CPU.

Suffix arrays and suffix trees both feature $\mathcal{O}(n)$ space consumption. However, thanks to the lower overhead of storing an array compared to a tree, suffix arrays practically use far less memory than suffix trees, up to a ×20 factor depending on how the tree storage is implemented – which has an impactful gain when working on real-world large datasets. Re-

garding time search, for a query m letters long, suffix trees feature an $\mathcal{O}(m)$ complexity, whereas suffix arrays features an $\mathcal{O}(\log n)$ one, thanks to the algorithmic efficiency of the binary search. However, typical real-world applications within our context (*i.e.* long duplications search through seed-and-extend strategies) will typically feature huge numbers of searches of relatively small strings (typically in the range of the dozen of characters). In this frame, the asymptotic complexity is not as relevant as the corner case of small strings search, in which both suffix trees and suffix arrays seem to perform well enough. On a side note, this is expected, as the logarithm of the length of the human genome is $n \approx 21.9$ and that using k-mers of $m = 20bp$ are common; in this case, the asymptotical complexities are thus nearly equal. Thus, at genomic scales, the lower memory consumption of suffix arrays is a neat gain.

Table 2.6: An example search of all the occurrences of "ACA" in the reference string "MACACA" using a suffix array. A binary search (first column) is performed on the suffix array (third column), using as a comparison operation the lexicographical comparison between the reference query on the one hand, and the substring of the reference string starting at the considered position. The binary search yields the range [1-2] (second column), corresponding to the elements [3, 1] in the suffix array, pointing to the substrings of the reference query (`ACA` starting at pos. 3 and `ACACA` starting at pos. 1) starting with "ACA".

Comparison result	Index	Suffix Array	Corresp. substring in reference
<	0	5	A
=	**1**	**3**	**ACA**
=	**2**	**1**	**ACACA**
>	3	4	CA
>	4	2	CACA
>	5	0	MACACA

The last aspect to discuss is the spatial and computational complexities of the array construction. Although it is an interesting question and improvements are still being introduced[80], what really matters in our use case is that algorithms featuring linear complexity (in relation to the reference string length) in both computational and spatial complexities are available, which is indeed the case[90]. Obviously, any improvement is most welcome, but as in our use case, the construction time is relatively short compared to the total run time, advantages concerning the latter are, albeit welcome, not as relevant as advantages concerning the former.

2.4 Existing Tools

After this overview of the common foundations shared by many programs dedicated to duplications searching, let focus now on a tour of the existing programs, and detail why we developed a new solution for our use case in spite of the current offering.

2.4.1 Global Alignment Analysis

A popular method to deal with duplications is the analysis of global DNA material alignment with locally developed, roughly described and often unpublished programs. They all seem to follow the same general scheme. First, given a sequenced DNA strand, it is run through a local alignment software, that is designed to look for local similarities inside a DNA string by performing an alignment of the string with itself. Such programs include, among others, BLAST[3] and its variants[4, 21, 97, 81], and aligners like MUMMER[28], YASS[95], or LAST[67]. Then, the resulting data can either be plotted as a dotplot and be processed manually, or by a custom script to gather the found areas in similarity islands. This overview does not detail the differences between the many use cases requiring such a pipeline. One may be looking for gene families, traces of ancient chromosome duplications, degenerated pseudogenes, and so forth, resulting in many different constraints and technical choice. Such an example is the use of the `nucmer`[91, 27] alignment tool, shipped with the MUMmer tools suite[75] and that may be diverted from its intended use of genome alignment to detect similarities inside a given input.

WGAC is the most used, published [13], pipeline for SDs mapping. It first filters out common repeats thanks to RepeatMasker[119], before aligning the remaining sequences against themselves. From this alignment, it differentiates the unique DNA material from the duplicated one. Common repeats are then reintroduced to produce the final result.

But the bottleneck all these methods share is the need for an alignment of the input sequence. If this method can scale without too much problem with DNA strands up to a few hundreds thousands basepairs long, their squared asymptotical computational complexity make them fall flat at larger scales. Thus, these methods are not fit for our use case for this very reason, as we need a solution that may scale up to multiple genomes.

2.4.2 Database-Based Tools

RepeatMasker[119] and its improvements[16] or wrappers[22] have another strategy. They call upon experimentally established database of known repetitions of many scales, then implement an approximate string matching algorithm to locate exact or slightly degenerated occurrences of the sequences contained in the database in the concerned genome.

Outside of any technical concerns, the main reason we cannot use these tools for SDs mapping is that they only work as well as their database is exhaustive. But as we wish to map duplications *de novo*, *i.e.* without any prior knowledge on them outside of the constraints we set on their length and identity rate, we cannot restrict ourselves to working with duplications similar to the ones that are already known. In fact, these kinds of tools are not designed for duplications discovery; their main intended use case is to mask *known* and *low-signal* repetitions from studied DNA to avoid confusing or slowing down other tools, further down the processing pipeline.

2.4.3 NGS Data Processing

Another class of tools and methods is designed for the search of duplicated areas from raw sequencing data[13, 12, 31, 124, 9]. As previously detailed, NGS technologies typically produce humongous numbers of overlapping small *reads* of the sequenced DNA strand. The average number of reads covering a given position in the final assembled genome, the *coverage*, generally ranges in the few dozens for an ideal sequencing run. However, duplicated sequences will tend to confuse assembly programs, leading to highly repetitive areas being underestimated, misassembled and merged together in smaller areas than the real ones, but exhibiting a disproportionally high coverage.

Several programs, some of them typically delivered as part of the sequencing machines and their processing pipeline, can exploit this information to detect the repeated areas. However, they can only detect highly repetitive areas, with a period of repetition of the same magnitude than the reads length and are rather imprecise for various reasons. First, they indirectly work at the read scale, around 100bp. Second, they typically implement statistical methods based on the variation of the coverage signal: if high variations can realistically be safely detected and attributed to repetitions, it is hard to determine whether variations of smaller amplitude should be attributed to repeated areas with a lower repeat count or to other artefacts during the sequencing process. Conversely, low copy-number repeats will only produce a weak signal and thus may easily go undetected. Finally, once they have detected duplications, it is hard to map them to their actual position in the genome. Last but not least, they require access to the very large dataset of raw sequencing data, as well as to computational facilities dimensioned to the processing of this quantity of data. Naturally, although they do not fit our own use case, these tools r canonical use, *i.e.* helping in the assembly of the sequenced genomes.

The most used published pipeline, WSSD [12] locally align all the original reads against the final assembled genome, and search for regions exhibiting a significantly higher coverage than the rest of the genome. These regions are then deemed as putative SDs. Its complexities, both computational and spatial, are obviously huge; moreover, it requires an access to the original sequencing reads, a typically very large and unwieldy dataset.

2.4.4 Short Repeat Searchers

The tools we put in this category are designed to detect repetitions falling in the first partition of the problem space we proposed: short, numerous repeats, sequential or close to sequential. Although they do not target the duplications we are looking for, we still mention at least some of them, if only for their ubiquity or for the originality of their approach. Moreover, even if they can not be used for the direct detections of the SDs we are looking for, they may prove useful later on. Ineed, as we previously noticed, SDs – especially in primates – come in several classes and contain different types of genetic material. Thus, being able to identify the content of micro-repetitions inside these macro-repetitions is very helpful when considering

SDs from a biological perspective.

Tandem Repeat Finder (TRF)[17] is one of the first tool developed for *de novo* generic detection of tandem arrays, or tandem duplications. It does not require any previous information on the period, count, or content of the repetitions to detect, and work under the Levenshtein distance – although it is not explicitly mentioned in the accompanying paper. It works by sliding a window across the reference DNA string to find exact matches close enough from each other to be considered as duplicons. It then makes use of a probabilistic modelization based on a Bernoulli distribution of mismatches between duplicons to find the non-exactly matching duplicons. However, its ability to detect sequential repeats according to the Levenshtein distance is also its curse, as it leads to a large complexity of the algorithm, making it unpractical for the detection of repetitions with large units.

mreps[70] features a better computational complexity allowing it to run on larger sequences, but at the cost of only being able to detect repetitions according to the Hamming distance, *i.e.* without neither insertions nor deletions between the duplicons, only substitutions. It finds all the duplicons matching with up to k mismatches, then applies an heuristic processing on this dataset to find the most fitting period, and then filters out the statistically insignificant repeats. The process is repeated for all k up to a user-set upper boundary K [3]. The results of every iteration are then merged together to form the final result.

Spectral Repeat Finder (SRF)[113] takes a novel approach, in which the studied DNA string is considered as a time series, *i.e.* a discrete signal. From this perspective, SRF computes the power spectrum of its discrete Fourier transform over a window sliding along the input string. In these spectrograms, noticeable peaks might point to repeat with a number of basepairs per duplicon equal to the corresponding period[4]. From there, a closer look at the sequences that triggered the peak detection with an heuristic method will determine the repeated unit within these windows. Not only is this method very easily parallelizable (both on CPU and GPU), but it also offers an elegant solution to finding repeated areas under the Levenshtein distance (rather than the more constraining Hamming one) thanks to the permissivity resulting from the mapping of the problem from the discret space of stringology to the continuous one of signal processing. Some drawbacks are the quick fading of the signal for repetitions with low copy counts, the difficulty to realistically work on very large windows, and the inability to detect non-tandem repeat, whose signal would appear aperiodic.

PRAP[22] is designed to detect repetitions in prokaryote genomes. Rather than developing a whole new tool from the ground up, the authors designed a convenient wrapper around

[3] That should be quite small, as the kind of repetitions searched by mreps typically ranges in the dozens of basepairs per duplicons.

[4] Although a repeat of enough duplicons will always result in a peak, the reciprocal is not true.

several existing tools: MegaBLAST[143], RECON[78], VisCoSe[120], RepeatMasker[119], and Artemis[106]. PRAP is designed to work on prokaryotes genomes and so is tuned to their peculiarities. It uses an approach that could not scale to the dimensions of the primates genomes concerned by our project (especially due to the necessity of MegaBLAST-ing the studied genome against itself). However, it highlights an important point. A tool should, besides its technical capacities, always take care of offering a convenient handling to its users and be interoperable with its surrounding ecosystem.

Red[46] is another original approach: it uses machine learning to detect highly repetitive areas in a reference DNA string. Schematically, Red works in three phases. In the first one, a score is assigned to each nucleotide in the reference query, equal to the numbers of occurrences in the genome of k-mers identical to the one starting at its position[5]; the signal obtained over the whole genome is then smoothed using a gaussian blur, *i.e.* a convolution of a gaussian curve. In a second phase, the resulting signal is used to mark each nucleotide as either being, or not, part of a duplicated area depending on the local smoothed score. A HMM automaton featuring four outputs (high/low probability of the current nucleobase to be in a non-repeated/repeated area) is then fed this crude segmenting to train on. In the third phase, the automaton is run on the whole genome to give a better, more precise segmenting of the sequence between repeated and non-repeated areas. Akin to SRF, this method allows the detection of repeats under the Levenshtein distance, thanks to the translation of the problem to a continuous space. On the upside, Red is very fast and massively parallelizable; and when compared with other methods, it offers excellent results according to the author's benchmarks. On the downsides, it relies on a high count (to form a statistically significant peak in the signal) of spatially close (for the signal to live through the blurring operation) duplications for an effective detection.

Similarity Chaining-Based Approaches many other tools have the same goal, *e.g.* PILER[34], DUST[89], etc. However, they all display the same conceptual bottleneck that prevents their use at the multi-genomic scale. As they rely on the exploration of the results of a local alignment program, they display an indirectly high complexity, penalizing their global runtime by too much to be realistically usable for large genomes studies.

2.4.5 Long Duplications Searchers

In the previous section, we have described tools that offer a good overview of the field, either for their ubiquity, their legacy, their large use or the innovativeness of the approach regarding the detection of high frequency, low amplitude, high repeat count regions in a given DNA string.

[5] k is typically chosen in the order of magnitude of the dozen.

Let us now focus on programs designed to work on the subset of the solution space of duplications finding relevant to our study, *i.e.* the detection of large amplitude, irregular, low repetition count and degenerated repeats. To the knowledge of the author, the published large-scale studies dedicated to the segmental duplications (or other large duplications)[19, 13, 12, 24, 23, 114, 10, 115, 85, 45, 144] never used a pure digital approach on the assembled studied genomes. Methods used are typically statistical analysis of the reads coverage during assembly (WGAC[13] and WSSD[12]), *in vitro* methods (and the consequent low resolution), whole genome comparison, either with BLAST and its derivatives or by dotplots, or a combination of all of the above. But although there are not many of them, some tools have been designed to detect long, sparse repeats.

Long Repeat Finders This family of programs is conceptually very close to the aforementioned alignment-based ones; their main difference being that they target longer duplications and are thus processing local alignment results according to this goal. A representative member of the long repeat finders programs family, OSFinder[50] is designed for the detection of orthologous segments among chromosomes or genomes. It follows a four steps algorithm: first, anchors, or seeds, *i.e.* short, nearly identical, orthologous segments must be fed to the algorithm. They can be found by BLASTing the reference query against itself or a target, by manually finding orthologous genes among the studied sequences, or other similar methods let at the discretion of the user. In a second time, these anchors are processed to detect the collinear ones, *i.e.* those following a similar distribution in the same direction. They are then chained together, before being output as orthologous segments.

Other programs from the same family, *e.g.* ReD[6][6](not to be mistaken with the aforementioned Red), DAGchainer[49], RECON[15], or Cinteny[117] follow a similar global scheme: chain together in larger regions the seeds provided by another program. Among them, the peculiarity of OSFinder is the use of a Markov chain model to automatically discriminate really orthologous segments from "accidentally" similar segments, in order to avoid an hypersensitivity of the results to algorithm settings defined by the user, which would result in imprecise results. Both orthologous segments and tandem gene arrays are close to SDs in our classification of duplications, they only exhibit typically lower identity rate and repeat counts. So the mapping of SDs with programs designed to detect these kind of repeats would theoretically be possible.

However, a common constraint of these families of programs (although we only mentioned a few of them, it remains true for the others the author knows of) is that they first need to be fed anchors for the joining algorithms to work on. A first solution to find anchors is to use known shared elements among the duplicated units, *e.g.* orthologous genes or known preserved sequences. A second one is to perform a BLAST (or another aligner) of the query sequence against itself, and use its result as anchors. But none of these solutions is usable in

[6] ReD is aimed toward the detection of tandem gene array rather than orthologous segments, but the two tasks are computationally extremely close.

the case of *de novo* SDs mapping; the former because it would require *a priori* knowledge concerning the content of the SDs to be found, and the latter because if running BLAST on a relatively short genome (*e.g.* the *A. thaliana* genome mentioned in ReD publication) is easy, doing so between several mammals genomes would not scale nicely. Therefore, we can not use these families of programs in our use case, as they either require discouraging computing times or *a priori* knowledge concerning the searched duplications.

Vmatch[135] is an unpublished, recently open-sourced, set of programs whose manual is available on the author's website. It is designed – among other tasks irrelevant in our context – for the detection of similar matches among sequences, and their subsequent clustering in duplications families. Vmatch includes algorithms from several older tools (that we thus will not detail individually here), including but not restricted to REPuter[74] and RepeatFinder[139], respectively used for the matches detection, and for their subsequent clustering in matches families.

In a typical workflow of SDs mapping, two subsystems of Vmatch would be used. First, the sequence would be searched for high similarity, non-exact matches thanks with the successor of the REPuter algorithm included in Vmatch. Then, this long list of two-armed fuzzy matches would be clustered together using the algorithm described in the RepeatFinder paper.

The REPuter algorithm for detection of highly similar matches in a reference string first starts by finding exact maximal repeats, using a suffix tree of the input string. Although REPuter can extend exact matches under both the Hamming and Levenshtein distances, only the latter is of interest to us due to the presence of indels in SDs duplicons. To this end, REPuter uses an algorithm featuring an $\mathcal{O}(n+z \times k^3)$ (n being the length of the input string, z the number of seeds and k the maximal number of errors) computational complexity. It extends both ends of each seed, while ensuring that the local alignments of these extensions stay under a k errors threshold. This computational complexity may remain tolerable on large datasets as long as the identity rate stays very high (and so, k^3 stays low) and the number of matches to extend is not too high compared to the length of the input string.

Once these matches are found, they have to be clustered together for two reasons. First, SDs are often found in families of more than two elements; and this grouping carry crucial biological information that would be lost if they were to be only reported in pairs. Second, assuming a k set to 100, SDs featuring larger numbers of errors would not be found. Or, given that SDs are defined with an identity rate proportional to their length and not with an absolute number of errors, longer SDs will typically exhibit larger total number of errors: if a 1,000bp long SDs family cannot contains more than 100 errors between its duplicons, a 100,000bp one could display up to 1,000 errors between its duplicons. Hence, longer SDs will be cut in several smaller pseudo-SDs, that will need to be clustered in their larger actual parent.

According to its documentation, Vmatch uses an algorithm based on RepeatMasker,

which is built around four main steps. First, all the matches fed to the algorithm are sorted, first according to their first coordinate, then their second. Then, all matches either overlapping or closer to each other than a given threshold are merged together in a virtual segments table. In a third time, the matches are attributed class identifiers that are then echoed on the virtual segments they are part of, these classes being merged if they are involved in the same virtual segment. Once this third step is done, the fourth and last one is to BLAST all the virtual segments against each other, merging again classes exhibiting a close enough E-value. Outside of the BLAST step, this algorithm has an asymptotical computational complexity of $\mathcal{O}(p^2)$, p being the number of matches. But given that the number of matches is arguably linearly increasing with the length of the input string, the computing time is expected to asymptotically grows roughly in the square of the input sequence length. Combined to the asymptotically lower ($\mathcal{O}(n)$), but still practically present computing time of the BLAST step, it adds up to a very time-consuming algorithm.

Vmatch was the most promising tools of all the ones we tried, and worked well on small datasets (*i.e.* up to a few dozen of millions of basepairs). However, if the program used to search for matches continued to scale up without much difficulty on larger datasets, the one clustering the matches found in duplications families soon showed its limit due to its computational complexity.

Moreover, none of the Vmatch sub-tools used in our pipeline was implemented in a parallel manner. Therefore, we were not able to make the best use of our computing resources, as its memory usage prevented running more than a few instances of Vmatch on different datasets at once, resulting in a dramatic underuse of available CPU power. Therefore, we decided to develop our own program to tackle the problem at hand.

2.5 Objectives

2.5.1 Computational

As seen previously, the tool fitting the best the problem of the large search space of duplications mapping in genome (Table 2.7) proved to be Vmatch. However, although it displayed no marked difficulty to pre-process large datasets, its clustering abilities – a feature required for our study – proved to be too weak. Moreover, Vmatch is relatively memory-hungry and is not able use efficiently modern CPUs, as it is capable neither of multi-threaded nor multi-process operation.

We therefore decided that the best course of action was to develop a new tool that would be designed for the detection of reasonably degenerated, long, non-tandem repeated sequences, taking into account the past experiences and satisfying the trifecta of:

- *de novo* duplications detection, without any *a priori* knowledge on the duplicons to detect;

Table 2.7: Summary table of the projection of the considered tools characteristics on our requirements.

	Target	*Ad-hoc solutions*	*RepeatMasker et al.*	*TRF et al.*	*mreps et al.*	*SRF & Red*	*OSFinder et al.*	*Vmatch*
Target Sequences								
size	**1kbp – 1Mbp**	✓	✓				✓	✓
repartition	**Scattered**	✓	✓				✓	✓
degeneration	$\geq$ **90%**	✓	✓	✓		✓	✓	✓
de novo	**Yes**	✓		✓	✓	✓	✓	✓
clustered	**Yes**							✓
Algorithm								
comp. complexity	**Genome scale**		✓		✓	✓		
mem. complexity	$\propto \mathcal{O}(n)$		✓	✓	✓	✓		✓
parallel	**Yes**					(✓)		

- ability to work efficiently at genomic scales, with the implied use of parallelization and distribution;
- tunability to multiple search criteria, depending on the precise use case.

We also wanted this future piece of software to be easy to handle and transparent for the users, *i.e.* it should “just work”, not be cumbersome to use, and it should blend in their pipeline without hassle. This implied two main design concepts.

On the one hand, from an ergonomic point of view, the program has to be easy to use, and well documented, providing examples and a clear manual. When used, it should give a straightforward return on the current process, what was done, or what error happened.

On the other hand, from a technical perspective, it should use common formats for input and output, and should thus be easily integrated in any pipeline. And last, it should be reliable during runs, so that the user would not wait pointlessly for several hours to discover the run being thrashed by a memory access fault, a concurrency error, or other mishaps.

2.5.2 Biological

Segmental duplications are supposed to play a large role in the creation of variation in primate genomes, thanks to their peculiar layout and composition. But their role is not limited to variation creation. Indeed, studies of the human Y chromosome strongly suggest that they

play a role in preserving the integrity of major genes implicated in reproduction, that would otherwise be threatened by the absence of homologous recombination on the Y chromosome for lack of a sister chromosome. However, study of SDs is currently limited by the lack of adapted sequencing technology that would be able to sequence without ambiguity these high identity, strongly repeated sequences.

From the review of the promising new generation of sequencing technologies, it is expected that newly, precisely sequenced long fragments of previously nearly inaccessible parts of the genome will start to flow in large quantities, thanks to the ever-decreasing price and difficulty of sequencing, combined with the potential discoveries that will not miss to be fueled by this newly-found trove of data.

It is accepted that SDs play a major role in the dynamics observed on the human Y chromosome. The gene conversion happening between them acts similarly to homologous recombination, and with the same consequences: giving the haploid Y chromosome the opportunity to create variation at the individual level, while preserving its integrity at the species scale.

From there, we want to determine whether this phenomenon is observable in the other species close to the human. Is a bias toward SDs enrichment systematically exhibited in XY chromosomes? If so, are the ZW chromosomes, from the eponymous sex-determination system, sharing the same bias? As the ZW sex-determination system exhibits a similar karyotypic differenciation between males and females (albeit reversed), it might be subject to the same constraints, and thus have a similar solution. Finding such enrichment in SDs in other haploid chromosomes would be a strong hint towards NAHR and gene conversion among SDs playing a crucial role as a substitute to recombination in various evolutionary mechanisms.

To this end, we intend to expose the SDs content of the currently sequenced sex chromosomes publicly available in databanks, as a first step towards answering this larger question.

Chapter 3

ASGART

3.1 Introduction

ASGART (*A Segmental duplications Gathering and Refining Tool*) is the new program we developed [30]. It is designed from the ground up for the detection of long, sparse duplications, and aims to be as easy to use and as user-friendly as possible, all the while leveraging efficiently the underlying hardware. The challenges faced were manyfold.

The first one, naturally, is to develop an algorithm that answers the biological problem at hand. This requires a thorough understanding of the problem, from which an algorithmic solution can be built.

The second one is to implement the designed algorithm in order to take advantage of the available computing resources. The current trend in CPU evolution is for cores number to increase, while the individual performances of these core tend, comparatively, to improve slower than they used to. This trend is attributable to many factors, but the main ones are (i) a need to keep the thermal enveloppe small enough to be dissipated; (ii) the restrictions in size imposed by the need to keep the chip in sync; (iii) the increasing technical difficulties in thinning the engraving width of these devices. Therefore, the best current bet to develop a piece of software that must be able to efficiently use current and upcoming chips is to design parallel algorithms. Moreover, modern supercomputers generally tend to follow a clustered architecture, where multiple compute nodes are unified behind a single software point of access, and parallelizable programs will generally be able to use more of these nodes at once, thus reducing the effective computing time.

Third, interoperability and ergonomy are a must. It is important to ease the work of the final users by abiding by the standards of the field regarding input and output formats, so that programs can be chained together in compute pipelines. The more seamless the process is, the more the user will be able to focus on its actual work instead of struggling with format conversions. Also, a tool should ideally feature a low entry bar, letting users use it without hassle, while still letting more possibilities available through advanced options.

Last, data exploitation and exploration are a challenge in themselves. With the very large

dataset typically featured in genetic study, especially at the sequence level, a simple analyse by a human alone is nearly impossible. Therefore, programs should provide a way to display their results to the user, in a format adapted to the problem at hand.

3.2 Algorithm

Let us now focus on the algorithm we devised to answer the problem. The core concept behind ASGART's algorithm is simple. Given a minimal identity rate and length characterizing a segmental duplication, there is a minimal length such that two substrings of this length in each of the duplicons exactly match (Figure 3.1).

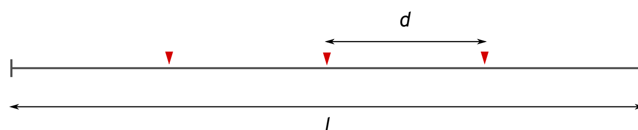


Figure 3.1: Illustration of the worst possible case for mismatches repartition, where mismatches (represented as red wedges) are homogeneously laid out in the duplicon. For a minimal duplicon length of l bp and h identity rate, the worst spreading case is one mismatch every $d = (100 - h) \times \frac{l}{100}$ bp on the duplicons.

For example, given a minimal length of 100 bases and an identity rate of 90%, it is guaranteed that there is at least a 9bp long substring common two by two between the units the segmental duplication. However, in actual cases, dissimilarities between repeated units tend to be grouped in clusters of indels, and SNPs are seldom homogeneously distributed among repeated units. Thus, ASGART's strategy is to, first, gather duplications between two fragments by looking for subsequences from the first fragment that exactly match other fragments in the second subsequence. It then clusters them together to find the repeated units composing the SD, according to the scheme detailed below:

1. pre-process DNA fragments in an efficient data structure;
2. gather identical k-mers from the two fragments;
3. merge and cluster these identical substrings together to form families of segmental duplications.

3.2.1 Definitions

Before delving deeper in the algorithm, let us set up a few formal definitions to present the kind of data ASGART will be working with.

We define a **string** S as a sequence of letters from an alphabet Σ. For instance, the alphabet for DNA is $\Sigma_{\text{DNA}} = \{\text{A}, \text{T}, \text{G}, \text{C}, \text{N}\}$ [1]. The ith letter within a string S is denoted $S[i]$.

[1]The N letter is used to denote a base that the sequencing process could not determine.

A **substring** – or segment – from the ith to the jth letter of a string S is denoted $S[i, j]$.

The **identity rate** between two strings is defined as the 100-complement of the ratio between the Levenshtein (or edit) distance of these strings and the length of the shortest string. For example, two 100 bp long strings differing at ten loci have a 90% identity rate.

A **duplications family** of length l and minimal identity rate h is a set of strings at least l bases long, so that each has an identity rate of at least h with the others: $SD = \{S_i, h | i \in [\![1; n]\!]\}$ is defined by the n repeated units – or arms, or duplicons – S_i and the minimal identity rate h between each pair of repeated units.

The **distance between two sets of segments** $SS_1 = \{SS_{1_i} = S_1[a_i, b_i], i \in [\![1, n]\!]\}$ and $SS_2 = \{SS_{2_j} = S_2[a_j, b_j], j \in [\![1, m]\!]\}$ is defined by

$$d_{SS}(SS_1, SS_2) = \min_{i,j}(d_S(SS_{1_i}, SS_{2_j})) \tag{3.1}$$

where the distance between two strings S_1 and S_2 is defined by

$$d_S(S_1, S_2) = d_S(S_1[a_1, b_1], S_2[a_2, b_2]) = \begin{cases} \max(0, a_2 - b_1) \text{ if } a_2 > a_1 \\ \max(0, a_1 - b_2) \text{otherwise} \end{cases} \tag{3.2}$$

It can intuitively be understood as the either 0 if the considered segments are overlapping or contiguous, or the number of basepairs between the two of them otherwise.

3.2.2 Pre-Processing

As inputs, ASGART takes:

- two DNA fragments, denoted A of size A_{size} and B of size B_{size};
- a probing size p_{size};
- a maximal gap size g_{size}.

ASGART features other options, but they are irrelevant to the core algorithm. Both p_{size} and g_{size} influence the granularity of the results, and must be set by the user according to the characteristics of the duplications they are looking for, namely their length and their minimal identity rate. The two input fragments are actually an abstraction due to the algorithm internal methodology, so *e.g.* if the user wishes to look for duplications inside a single DNA fragment, the two fragments A and B will actually be identical. Similarly, if the user wishes to look for reversed and/or complemented duplications, ASGART will proceed as if the two input fragments were the input fragment and its reversed and/or complemented self, though a single FASTA file is used.

As output, ASGART gives a list of duplicons composing the duplications families spanning the two fragments, in the output format chosen by the user.

At first, a suffix array of B is created thanks to the `divsufsort` library[90] – which is, according to our trials, the most performant one currently available.

In order to trade memory use for performances, we build an index of the positions of all the possible 8-mers in the suffix array. Thus, any binary search ASGART will do starts directly in a partition of the suffix array a 256th the size of the whole array, for a memory cost of ~6MB, that we deem negligible compared to the improvement in performances. We chose 8-mers because they can be very easily converted to and from 64 bits words with a simple operation of memory reinterpretation, and thus are very easily hashed with this simple method to index the cache.

3.2.3 Clustering and Gathering

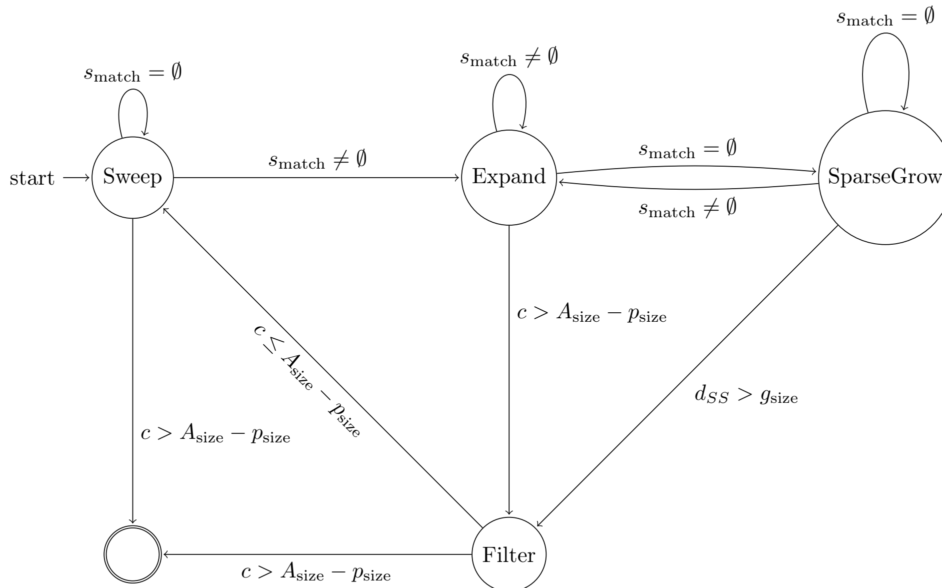


Figure 3.2: ASGART finite-state automaton schematic representation.

The gathering of high similarity zones and their subsequent clustering in SDs is the heart of ASGART. From a suffix array of B, a finite state automaton (detailed in Figure 3.2) will scan A, store and merge identical k-mers among the two fragments, and position and cluster them in proto-SD according to the probing size and the maximal gap length provided by the user. The automaton contains four states: *Sweep*, *Expand*, *Sparse Grow* and *Filter*. An illustrated example of our algorithm is shown in Figure 3.3.

ASGART is implemented as an infinite loop, where the code of one of the state is executed at each iteration. The executed state is set in the global variable `state`, that each state must thus update depending on the transition condition. We will now detail the pseudo-code found in each of these states. Please note that for clarity purposes, the pseudo-code illustrating each of these states does not include DNA strands length checks and other similarly technical details.

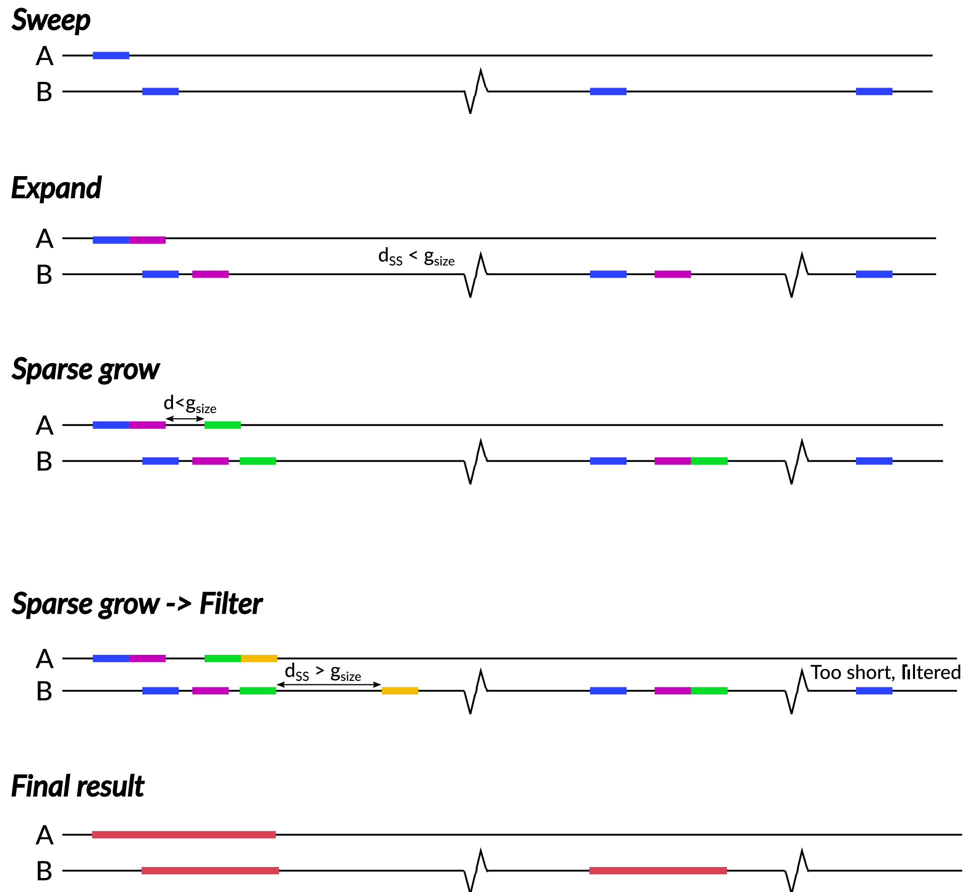


Figure 3.3: A schematic example of an ASGART job. Each differently colored segment is coming from a different step in the automaton. (1) After a first successful *Sweep*, a k-mer in A has three matches found on B. (2) A second segment on A, contiguous to the first, has two matches on B close to the existing ones. Those are not contiguous to the existing matches from the previous step, but they are still close enough to be kept. (3) No contiguous k-mer to the last one on A having matches close enough to current matches set on B, the automaton switches to *Sparse Grow* state, until it finds one having matches on B close to the current set. (4) After having spending some steps in *Sparse Grow*, a match on B for the k-mer on A is eventually found. But it either does not have a match close to the current set or it is too far away; therefore, the automaton switches to the *Filter* state. (5) After the *Final* state, the third and last set of matches on B has been discarded, as it is too short with regards to the settings defined by the user. The other set of matches is marked as being duplications of the part on A.

Setup

At the start, a few global variables of the automaton are set up and initialized, then the automaton is started in the *Sweep* state.

```
GLOBAL c ← -1               # So that we start at A[0] instead of A[1]
GLOBAL matches ← []         # The current set of matches
GLOBAL proto_sds ← []       # The progressively final list of duplication families
GLOBAL state ← Sweep        # The state in which the automaton is
```

The parameters set by the user are `g_size`, the maximal gap size, and `p_size`, the probing length. Some helper functions will be used in the pseudo-code, namely:

`d_SS(x, y) :: Set Segment → Set Segment → Integer` is the implementation of the aforementioned distance d_{SS} (Equation 3.1);

`filter(duplicons) :: Set Segment → Set Segment` takes a set of duplicons as its input, removes the duplicons not satisfying the criteria set by the user, and returns this result.

`search_for_kmer(kmer, target) :: String → String → Set Segment` uses the previously mentioned suffix array to return a set of all the k-mers in `target` exactly matching the argument `kmer`;

`merge_matches(currents, news) :: Set Segment → Set Segment → Set Segment` depending on a runtime argument, this function behaves differently. When ASGART works in non-interleaved mode (the default one), *i.e.* assumes that there are no nested duplications, `merge_matches` iterates over the segments in `news`. For each given segment new_i in `news`, if theres is one segment current_j in `currents` close enough to it (*i.e.* $d_{SS}(\text{new}_i, \text{current}_j) < g_{\text{size}}$), both are merged and current_j is replaced by $\text{current}_j \cup \text{new}_i$ in `currents`. When in interleaved mode, `merge_matches` also iterates over `news`. If the current segment has a close neighbor in `currents`, the behavior stays the same and they are merged together. But otherwise, the current segment will be appended to `currents`, with a tag branding it as a sub-duplication families starting at the current position of the cursor instead of being ignored.

Sweep

In the *Sweep* state, the automaton will move a cursor c along A, while probing B for k-mers p_{size} bp long identical to the one starting at c on A. If a non-empty set of substrings $\{B[x_i, x_i + p_{\text{size}}], i \in [\![1, N]\!]\}$ matching $A[c, c + p_{\text{size}}]$ are found, this set of matches s_{match} is stored and the automaton switches to the *Expand* state.

Otherwise, the exploration continues by increasing the cursor value until either it finds a non-empty set of matches or reaches the end of the fragment. This state corresponds to the search for the first matching parts of a duplicated family repeated units.

```
SWEEP:
c ← c + 1
matches ← search_for(A[c, c + p_size], B)
if matches not empty then
    state ← Expand
else
    state ← Sweep
end if
```

Expand

In the *Expand* state, the automaton expands a potential proto-SD detected during the Sweep state. The cursor c traverses A while the distance d_{SS} between the set of the matches in B of the probing k-mer and the current set of matches s_{match} is smaller than g_{size}. Matches found are successively merged in s_{match}. When there are no more matches of the current probing k-mer on B, or if the distance d_{SS} to the current set of matches is greater than g_{size}, the automaton switches to the *Sparse Grow* state. If the end of A is reached, the automaton switches to the *Filter* state. This state is dedicated to the gathering of either exactly matching portions of future duplications or handling deletions in the repeated units on B.

```
EXPAND:
c ← c + 1
new_matches ← search_for(A[c, c + p_size], B)
if d_SS(matches, new_matches) ≤ g_size then
    matches ← merge_matches(matches, new_matches)
    state ← Expand
else
    state ← SparseGrow
end if
```

Sparse Grow

The *Sparse Grow* state is reached when the current set of matches is not immediately expandable. The cursor c will traverse A until a non-empty set of matches for the current probing k-mer is found in B. If the distance d_{SS} between the matches of the probing k-mer and the current set of matches is smaller than g_{size}, the sets are merged and the automaton switches back to the *Expand* state. The automaton switches to the *Filter* state if the distance exceeds g_{size} or when the end of A is reached, *i.e.*, $c > A_{\text{size}}$. The *Sparse Grow* state also handles deletions in the repeated units on A.

```
SPARSE GROW:
gap_size ← gap_size + 1
```

```
if gap > g_size then
    state ← Filter
else
    new_matches ← search_for(A[c, c + p_size], B)
    if d_SS(matches, new_matches) ≤ g_size then
        matches ← merge_matches(matches, new_matches)
        state ← Expand
    else
        state ← SparseGrow
    end if
end if
```

Filter

In the *Filter* state, ASGART extracts a substring of fragment A and a corresponding set of matching segments of fragment B. These data are merged and filtered to ensure they satisfy the minimal length constraint set by the user. The automaton then switches back to the *Sweep* state if it did not reach A end.

```
FILTER:
proto_sds ← append(proto_sds, filter(matches))
state ← Sweep
```

3.2.4 Complexity

A major metric for algorithms geared toward large dataset analysis is their complexity. The computational (respectively memory) complexity describes the asymptotical behavior of the compute time (respectively memory) consumption as a function of the length of the input. Formally, given a function $g(x)$ that is non-zero for sufficiently large values of x, a function f is said to be $f = \mathcal{O}(g) \Leftrightarrow \lim_{x \to \infty} |\frac{f}{g}(x)| < \infty$. It can be computed for *e.g.* the worst and average case, and is a good indicator of how the use of resources scales in relation to the input data size.

It was critical that ASGART complexity was low enough to work properly at the genomic scale on common hardware. From a memory point of view, ASGART consumes non-negligible amounts of memory at some points. First, the reading of the input FASTA files is directly proportional to their size. Then memory is allocated to build the suffix array of the second strand, that is also directly proportional to its length. Thus, ASGART memory complexity is $C_{\text{memory}} = \mathcal{O}(n) + 2 \times \mathcal{O}(m) \sim \mathcal{O}(\max(m, n))$, where n and m are the respective lengths of the two input strands. ASGART memory use is thus linearly proportional to its inputs size, which is a favorable result. This result stems from the fact that ASGART needs to store the two input strands, as well as the suffix array of the second strand, whose

size is directly proportional to the size of the strand itself.

The second complexity indicator relevant to us is time complexity, that reflects the asymptotical growth of computation time as a function of the input lengths. ASGART probabilistic time complexity is equal to $C_{\text{time}} = \mathcal{O}(n \times (\log(n) + s^2))$, where n is the length of the largest input, and s the probabilistic cardinal of the matches set, *i.e.* the expected numbers of identical k-mers. This result is intuitively understood as a step in the *Sweep* state has a complexity of $\log(n)$ (for a binary search) and a step in the *Expand* or *Sparse Grow* state has a complexity of $\log(n) + s^2$ (a binary search plus the merging of the found segments). Each of them happening up to n times, the final complexity is $C_{\text{time}} = \mathcal{O}(n \times (\log(n) + \log(n) + s^2)) = \mathcal{O}(n \times (\log(n) + s^2))$

Obtaining a run time estimate from this formula is nearly impossible in a real-world application. Because of the impossibility to find probabilistic estimates on one genome, all the more all ot them, computing a theoretical approximation of s is out of reach. Let us consider, for instance, the genomes of the human and the zebrafish, that are roughly of the same size (respectively 3Gbp and 2.9Mbp). The most repeated 20-mer of the human genome is 20 × `A`, present 448,024 times. For the zebrafish, the most common one is a tandem, 10 × `TA` (and its *alter ego*, 10 × `AT`), occuring 2,686,477 times, so roughly six times more – although the zebrafish genome is shorter than the human one. As such measures can only be obtained *a posteriori*, we cannot practically refine further this complexity estimation.

So, when ASGART will search for duplications in genome containing duplications of a very-short period (a few bp) and made of tremendous number of duplicons, some k-mers will be present in absurdly high counts, and ASGART will stutter. We propose two solutions to limit this problem. The first one is run-time parameter allowing the users to set an upper limit to the number of matches for a k-mer, over which concerned k-mers are ignored by the automaton. The second one is, like advised by many other tools, to first mask the highly repeated subsets of the concerned dataset with *e.g.* RepeatMasker, Red or SRF, before running ASGART.

3.3 Implementation

First and foremost, ASGART[2] as well as its online version[3] are freely available under the GPLv3 license. After this overview of the algorithm, let us now examine what were the multiple technical choices we made to have an implementation that would satisfy the constraints as best as possible.

[2] `https://github.com/delehef/asgart`
[3] `https://asgart.irit.fr`

3.3.1 Constraints

Data Scale

First and foremost, ASGART must be able to work on data that will rise up to the genome scale. As we envision to explore segmental duplications with an approach combining both genomics (single species) and phylogenetic (multiple species) ideas, ASGART has to be able to map these duplications on sequences that may range from chromosomal sizes (a few millions basepairs) to genome-scale (a few billions basepairs). Moreover, as a phylogenetic approach obviously implies the comparison of several species, ASGART must be able to work seamlessly at the multi-genomic scale.

From a computing perspective, this constraint is the capacity to work on two input sequences, each of them susceptible to go up to the genome scale. The genome scale represents an order of magnitude of the gigabyte, accepting an encoding of one nucleotide by byte[4].

OS

Naturally, ASGART must run on any OS commonly used in bioinformatics settings. The most used are probably GNU/Linux distributions, typically RHEL or its free derivative CentOS on the computing clusters, and Ubuntu on the personal workstations. However, macOS tends to be quite popular on laptops, therefore ASGART must also be able to run on this OS. And although Windows' use tends to be dwarfed by these two others OSs, it is still not a negligible one and not making our software available on it might hamper its accessibility for quite a large number of potentially interested users. Therefore, we set down on the {GNU/Linux, macOS, Windows} triplet as our domain constraint regarding the portability of the language stack.

Hardware

ASGART should be able to use at its best any machine. Thus, it should make the best use of available hardware parallelism opportunities. This means exploiting multi-core CPU as well as being able to share a given workload among multiple machines.

However, the technologies used should be easily available or easily set up by novice users of the three main OSes (GNU/Linux, macOS, Windows) used in a bioinformatics context, so that ASGART can be usable without external assistance, and we offer precompiled binaries to reach this goal. On the downside, this constraint prevents us from using some common facilities such as *e.g.* MPI.

[4]A nucleotide could be encoded on only three bits (taking into account the sequencing uncertainty `N` nucleotide), therefore fitting two nucleotides in a byte. However, the gain in memory use was deemed unworthy compared to the increased time cost of reads in our current use case.

I/O Formats

Input One format is mainly used in bioinformatics for storing assembled DNA sequences, the FASTA format, that was born from the eponymous piece of software[82], one of the first alignment software to be developed. A FASTA file is made of a succession of paired headers and sequences data. The headers are made of one line starting with a greater-than sign, while sequences are stored as strings of `A`, `T`, `G` and `C` and other characters following the ASCII encoding, corresponding to the nucleobases of the sequence. However, there is no strictly formulated normalization for the structuring of meta-informations in the header. *De facto* sub-standards have emerged, based on the use of various separators and reference to external nomenclatures, but there is no general, standardized way to store meta-informations with the sequences themselves.

For instance, the presence or not of a space after the leading greater-than sign, whether a sequence string can be cut on several lines, whether blank lines are allowed, the semantics of upper- and lower-case nucleobases, etc. may differ in interpretation depending on the concerned program.

Therefore, the FASTA parser has to be permissive enough to acknowledge all of these corner cases, or at least fail with a precise error message, so that the user may at least understand how to fix their malformed input data.

Output The existing formats for storing genomic features are not many. The most widely used ones, by a large margin, are the GFF formats[43, 44]. However, there are two extensively used versions of these formats: the legacy GFF2 format, and the more recent GFF3. Although GFF3 is a superset of GFF2 and offers more functionalities, it has not yet replaced it due to the quantity of applications still producing and consuming this legacy format.

If the GFF formats are useful for interoperation with other bioinformatics tools, they are intrinsically a rather poor format from a computer science point of view. In a nutshell, they store one feature[5] and their properties per line in tabulation-separated fields, plus some headers at the beginning of the file. They also include a simple id/key mechanism to allow fields to refer to other ones, although with strong limitations on how it can be done while respecting the standard. However, in our case – storing the information relative to a family of SDs (which are represented as a group of features in the GFF format) –, there are no way to elegantly store data for the whole family.

Two possibilities exist to overcome this: either accept as a convention to store data concerning the whole family in only the first member, or repeat the information for every member. This flaw alone would not be a showstopper, but the real obstacle is the non-existence of this format outside of bioinformatics. As a computer scientist, one may like to use already well-established, dedicated tools to perform any kind of operations on a dataset, be it *e.g.* through jq[129] or SQL[133] queries, Python or other scripting language, data mining

[5] A feature is a noticeable part of a genome; that may be a gene, a satellite, an exon, etc.

tools, etc. But the GFF formats are often not supported in these tools, due to it being a relative niche compared to widely used formats such as to JSON, YAML, XML, etc.

Therefore, we decided to support three output formats: the GFF 2 and 3 formats for interoperation with the existing bioinformatics ecosystem, and JSON for interoperation with more classical computer sciences tools.

Input Data & Settings

As explained before, we need ASGART to find duplications in assembled DNA sequences without any *a priori* information, in order to avoid any bias in the features found. We are following an exploratory approach, and, therefore, we expect to work on newly sequenced data, for which only sequencing results and informations are available. Moreover, it is not established that SDs will be displaying common features across species, or at which point. So, a strictly *de novo* approach is required, as it would be unwise to rely on existing databases.

Moreover, such an approach allows for a better reproducibility of the obtained results, as the only variable parts are the input data and the settings chosen by the user. Consequently, ASGART has to be isotropic regarding the input sequences and should not use any of their metadata (*e.g.* known genes, methylation patterns, ...).

DNA strands As input, ASGART naturally requires the two DNA strands between which duplications will be searched. To look for duplications within a single DNA strand, the same strand is used twice. ASGART will automatically detect such a case and, while acting as if the two identical strands were logically distinct, still only read and instantiate it once, thus limiting the memory use and sparing I/O and parsing time that would be wasted if it were to store and process twice the same exact data.

The strands must be in the FASTA format. Although ASGART is relatively permissive on its formatting and accepts, for the sake of convenience, files that should not be accepted if the format definition was strictly applied, it is still recommended that users follow the FASTA format conventions for obvious interoperability concerns.

Settings As described above, ASGART has several settings that allow the user to determine the precision and the scope of the search. In addition to the fundamental, algorithmic parameters, we added some "service" parameters, that while not influencing the course of the algorithm, change the behavior of the program in terms of interaction with the system, for instance where to write result files, how many threads to use, and so on. Their detailed list is available in the documentation.

User Experience

Safety is a primary concern: the user must be able to trust his tools. To build this trust, we think that three principles should be followed.

First, the program must be thoroughly documented, its options clearly described, and examples of use for most common cases highlighted for the user. For a user who is not necessarily as familiar with the command line as developers may be, simple explained examples may go a long way to help them get started.

Secondly, a clear indication of what is going on (especially during run times that may span for dozen of hours), featuring progress indicators or estimation of remaining time, is always a welcome addition. This way, the users may get an idea of the adequacy of the computing power at their disposal to the task at hand, and they will not worry about a program that may seem frozen, but is actually just working in the background without any kind of feedback to indicate its current state. Also, the perspective assuming that the user has thoroughly read all the manual and understood all the intricacies of the program is making a disservice to both the user and the developer; the first one probably ending frustrated by his experience with the program, and the second one seeing the technically interesting tools developed ending up unused due to frictions in user experience.

Thirdly, the program itself must be safe, *i.e.* it must either run correctly to its end or fail with handled, explicit errors, and could go up to the length of proposing solutions to fix arising problems. In no way should it abnormally terminate on memory errors, concurrency issues, or other intrinsic problems rising from implementation errors.

3.3.2 Technical Choices

Language Selection

We settled on using Rust, a language backed by the Mozilla Foundation, designed as a safe, high-level, strongly-typed, high-performance language. Its strict compiler is designed to prevent whole categories of potential runtime errors at compile-time – mostly memory ones. Thanks to its use of the LLVM backend[6], its performances are virtually equal to C++ ones. In our own benchmarks comparing our first prototype, written in C++, and our first Rust version, both had similar performances. If Rust fundamentally offers only one really new feature (namely its memory management model), we were interested by the peculiar set of features it combined.

First and foremost, Rust is designed as a low-level, high performances language. To this end, Rust uses no runtime outside of its standard library; it uses LLVM as its compiler backend, therefore leveraging this high quality, highly optimizing backend to offer final performances in the same magnitude as C or C++. To ease interaction with existing codebases, it features a trivial FFI with the C ABI}, so it can easily link with or be linked with C, C++, etc. code objects following it.

Secondly, Rust provides a large set of features, usually encountered in functional languages. It features a strong type system, helping the developer to improve the safety of its

[6]Originally developed for the clang C/C++ compiler.

program. In addition to the guaranties stemming from such a system, it also allows idioms that are common in *e.g.* Haskell or Caml, such as sum & product type, or pattern matching and deconstruction as first class citizens. Similarly, closures, lambda function and functions themselves are first class citizens, meaning they can be manipulated like any other native object of the language.

Thirdly, Rust features a strict memory model. Thanks to its pervasive use of precise memory semantics and a thorough analysis of the lifetime of every constituent of the program, the compiler features a stage called "borrow-checking". This stage ensures the memory safety of the program, detecting many race conditions, use-after-free, concurrent modifications, use of reference outside of scope, use of uninitialized memory, etc. The strong guarantees offered by the borrow-checker may force to rewrite some part of the code to ensure that it won't be triggered by questionable code[7]. If it may be inconvenient during the development, it forces developers to think about the interdependencies and lifetimes of their structures, and eventually leads to safer programs without the runtime overhead of a tracing GC.

Last but not least, Rust ships with Cargo, an integrated build tool fulfilling several roles at once:

Build system although one can directly call `rustc`, the Rust compiler, Cargo will automatically build a project, from the compilation to the linking steps, in a single command. This will work on all the platforms officially supported by Rust, ensuring a trivially portable compilation system on these[8].

Dependencies manager all Rust dependencies of a program built with Cargo must be specified, and Cargo manage them automatically on a per project basis[9], thanks to a central repositories. Of course, one can still provide a custom location for a dependency, be it a git repository, a local directory, etc.

Test suite Cargo also provides an integrated testing system. When invoked, it will build the project and run all functions marked as test functions[10] and provides a report on successes and failures. These test functions are backed with numerous features from the standard library dedicated to testing, such as various types of asserting functions.

Settings

ASGART works on the two FASTA or multiFASTA files given by the user. However, it also accepts many other technical and circumstantial options.

The options relative to the core algorithm itself, namely the maximal gap size and the length of the probing k-mer. They are respectively set to 100 and 20bp by default. The user

[7] Or put this code in `unsafe` blocks, inside which the borrow-checker is disabled.
[8] Tier 1 platforms include GNU/Linux, macOS, Windows, both 32 and 64bits
[9] Thus not polluting the global space and removing the need for isolation mechanisms, such as Python `virtualenv`.
[10] Which are obviously pruned from release builds.

may also set the minimal length of the members of a duplications family members, which defaults to 1000bp. Another option lets the user set the maximal cardinal of a duplications family. Indeed, as was said before, genomes typically contain a large quantity of relatively small duplicated areas. Therefore, trying to explore dozen of thousands of potential duplicons actually limited to a few k-mers of some extremely common structure, like AT satellites or poly-A tails, is not only useless, but extremely costly. As SDs families generally amount to the low dozen duplicons, the user may wish not to lose time exploring putative proto-SDs numbering in the dozen of thousands.

Regarding the input, the user can reverse and/or complement one of the input files, to look for reversed and/or complemented duplications. One of the input may also be trimmed, when the user does not wish to scan it in its entirety – this feature being particularly useful to slice input data to share the workload among multiple compute nodes. For performances concerns, the user can set whether or not ASGART will skip masked sequences, and set the maximal numbers of matches to take into account. Given that the average SDs families cardinal ranges in the magnitude of the dozen, it is typically safe to ignore k-mers with more than one hundred matches.

Finally, the technical options include mostly classical housekeeping features: output format, output file naming, number of threads to use, etc.

These options are detailed in the documentation, and a list of them with some additional details is available when launching ASGART with the universal `-h`/`--help` argument.

Parallelization

As mentioned in the requirements, being able to exploit all the parallel capacities of the platform ASGART is running on was mandatory. A strong advantage of the ASGART algorithm is that it is massively parallel: as the input strings are only needed in read access, strand A can be cut in as many part as needed, and each of these parts can be processed independently by a single thread.

That was the course adopted: we use a thread pool (defaulting to as many threads as there are cores available, but this behavior can be altered by the user) that is fed with parts of the input data (typically, a million base pairs long), whose results are collected, then merged once all of the threads have finished, following a classical divide and conquer, or map & fold, scheme.

As will be detailed later, this way of parallellizing ASGART leads to an efficient scaling in performances following the number of used cores – which is expected, given that there are nearly no need for synchronization nor for memory exclusion mechanisms. Moreover, there is nearly no memory overhead, given that the only thread-local data are the coordinates of the duplicons being collected, whose size is negligible (typically a few kilobytes) compared to the input data and the suffix array of the B strand (generally in the order of the gigabytes).

Output

Independently of the output format chosen by the user, the data that will be saved are the same:

- the algorithmic parameters that were given to ASGART for the run;
- a map of the FASTA files given in input, *i.e.* a list of each fragment and their coordinates, so that other tools may map duplicons to DNA fragments without needing access to the original FASTA files, which may be too large to be conveniently used;
- a list of all the duplications found by ASGART. Each of them will be characterized by a list of its duplicons, each of them being described by its position, in which file, and its length in base pairs.

Although ASGART can write results to JSON, GFF2 and GFF3 files, JSON is used as the default output format. It is a format that is easily readable by a human, but also extremely easy to integrate in any pipeline, as mature parsers for JSON exist for practically every language under the sun. Last but not least, it can be efficiently compressed for storage.

3.4 User Interaction

3.4.1 Interface

ASGART ships with two different interfaces for the end user, a command-line one and a web-based one. The CLI program is intended for either quick run on personal computers, or for long, intensive ones on *e.g.* compute clusters or servers. The web interface is designed to let several users access concurrently a centralized computing unit. It is divided in two parts; a web interface allowing users to submit jobs and allowing administrators to monitor them on the one hand; and a worker queue consuming the jobs and running them where specified in their configuration on the other hand. Thus, the web interface can be deployed on a frontal node, while dispatching the actual computations in other machines, *e.g.* over ssh.

CLI

The canonical interface for ASGART is the CLI application. CLI applications offer the most interesting set of features for scientific computing. They are easily deployed, as one does not require a whole graphical library as a dependency; they can run on an headless server, which is a major advantage when using remote hardware, such as a computing cluster; and they can adapt their level of verbosity and interactivity depending on runtime options or outputs (console or pipe).

ASGART's CLI tries to be informative and helpful (Figure 3.4), with an output colorized according to the importance of the concerned information, a few details on the current state

of the program, and a progress bar allowing the user to follow the progression of the computation. It adapts its output depending on whether its standard output is a terminal or a pipe. So, it can be used as a pipe source without filling it with useless output, such as ANSI sequences for colors or animated progress bar. This feature lets it be transparently informative when being run interactively, while not polluting *e.g.* log files with useless informations when run in a job queue.

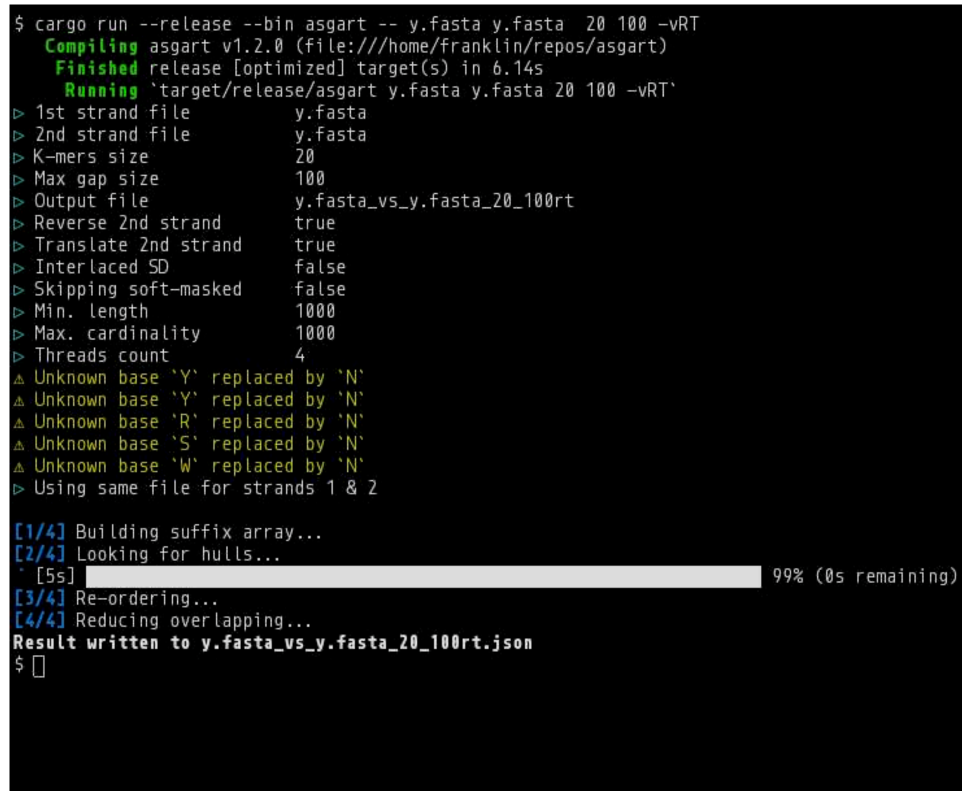


Figure 3.4: A screenshot of a successful ASGART run on the human Y chromosome.

Web Interface

The web platform is nowadays the most used solution to offer an easy access to a service. Compared to native applications, it offers a set of standard APIs that are available on every major platform, providing a consistent cross-platform experience to the user. And the ease to deploy and update applications thanks to the centralized nature of the application distribution process, as well as the easy access from the user perspective, are major advantages.

Therefore, we offer a web-based interface for ASGART. It runs on the BEAM VM, featuring the Elixir language and the Phoenix web framework. We had several motives to chose the BEAM platform. First, BEAM is a robust, mature VM, being used for decades; so using this technology is a safe bet concerning future development. Second, the BEAM VM is based on the actor model, and thus features many facilities for concurrent and parallel

programming that permeates through the languages targeting it. These features let us easily develop, for instance, our job-queue system whithout the need for third-party component. Finally, the BEAM VM is built as a resilient piece of technologies. Originally developed by Ericsson as the core of the Erlang OTP, it is designed around multiple agents communicating through message passing, such as one of them crashing does threaten neither the VM nor other agents stability. Thus, it is a solid platform well suited to web development, where a request failing or crashing does not hamper others.

This interface is, above all, targeted at non-technical users, and therefore features a job queue, mail notifications and the possibility to trigger execution on other machines than the one running the web server. It also features a basic visualization application in JavaScript (Figure 3.9) to allow users to quickly get a hang of the result of their run.

Figure 3.5: A screenshot of ASGART web interface current state.

3.4.2 Results Exploitation

Besides ASGART itself, a companion program is shipped with asgart for exploratory work on duplications families. It can plot or export subset of SDs mapping in various format and representation. From a JSON ASGART result file and the parameters set by the user during its invocation, an SVG graphical representation or a CSV tabular dataset of the SDs subset from the original file satisfying the mentioned restrictions is produced. The CSV output carries

the same information as the JSON output, and can be imported in spreadsheet programs such as Excel, Calc or Numbers. The SVG representation, however, is targeted at humans, to give a quick visual overview of the layout and other characteristics of the duplications. To this end, the user can also add a layer of interesting features contained in a CSV or GFF file to be incorporated in the output picture for visual study.

Flat Plotting

The first possible output format is the so-called *flat* plotting (an example is given in Figure 3.6). It is made of two parallel strips sitting atop of each other and representing the two input DNA strands. Each duplication is then represented as a line from a relative coordinate in the first strip matching its position in the first strand, to a relative coordinate in the second strip matching its position in the second strand, and whose thickness is directly proportional to the duplicon length[11] and whose color depends on if it is either reversed and/or complemented.

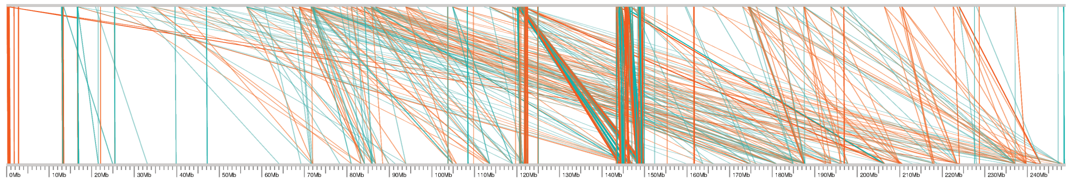


Figure 3.6: A flat plot of the human chromosome 1 SDs larger than 2kbp. Direct duplications are shown in orange, while reverse & complemented ones are plotted in petrol blue.

Chord Plotting

The chord-plot representation (an example is given in Figure 3.7) is, currently only available if ASGART was run on a single input sequence. It presents the input strand as a circle, and each duplication is represented as a Bézier curve computed from three points: a first point on the circle corresponding to the first duplicon position on the input DNA strand, the center of the circle, and a second point on the circle matching the start of the second duplicon on the input strand. As for the flat plot, its thickness and color depend on the length and orientation of the duplicon.

Karyotype Plotting

A third way of results visualization offered by ASGART is to project them on the karyotype of the multiFASTA file used as input (Figure 3.8). Obviously usable only when looking for duplications inside a single file, it lets the user quickly grasp the global positioning of the found duplications on a set of chromosomes, or other DNA fragments. Each fragment is vertically divided in two times two tracks, grouping, from left to right and in this order:

[11] Although a minimal width of a pixel is set to ensure small duplicons appear on the graph.

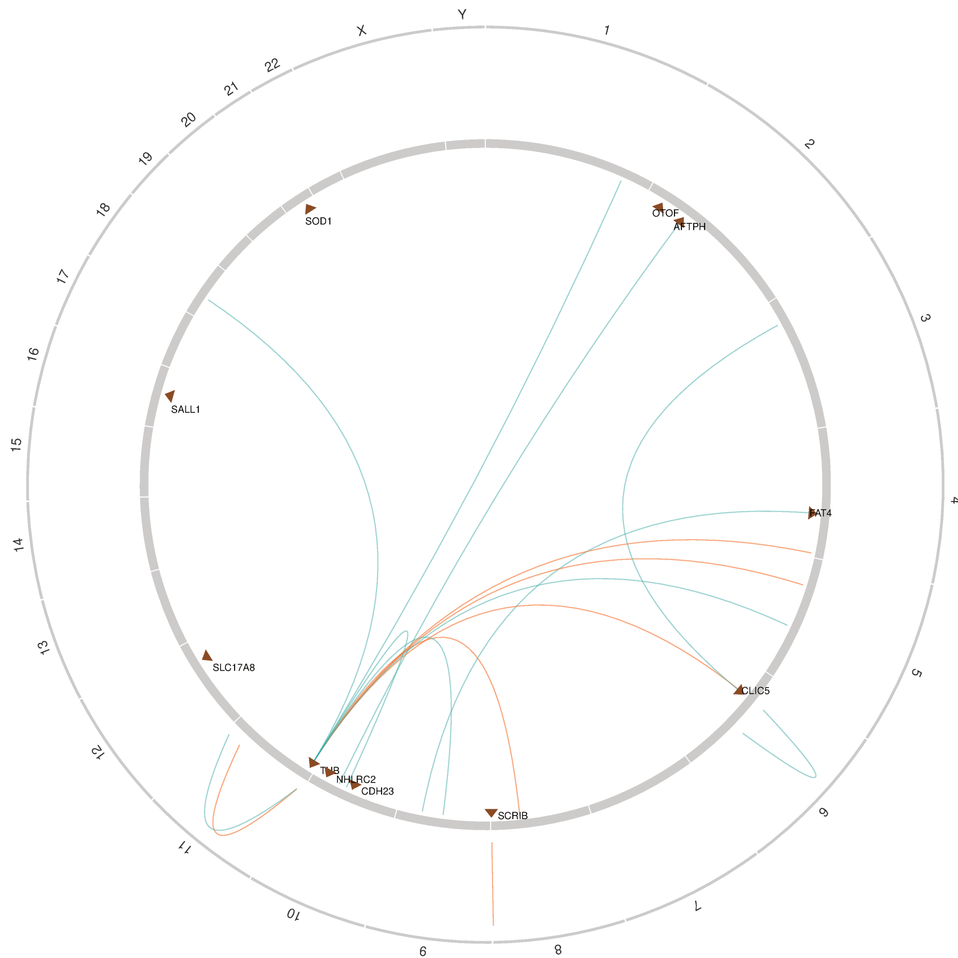


Figure 3.7: A chord graph representing human SDs close to a set of genes. Genes are represented by their name and a tick pointing to their position in the genome.

- intra-fragment, direct duplicon;
- intra-fragment, palindromic duplicons;
- inter-fragment, direct duplicons;
- inter-fragment, palindromic duplicons.

Karytoype plots are lighter in resource when opened, as they are only made of straight lines; they are faster and less-resource intensive to draw than the Bézier curves used to render the chord plots.

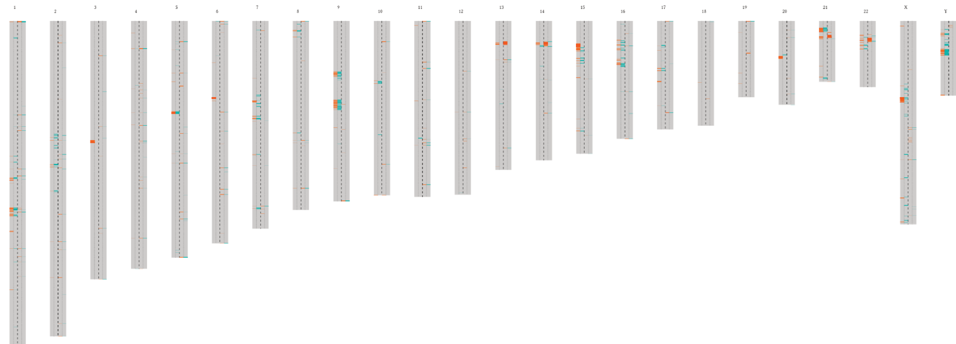


Figure 3.8: A graph using the *karyotype* type of plot to present a subset of the human duplications found by ASGART.

Web Visualization

The web application of ASGART also features an integrated viewer for a quick, online access to the results. Akin to the dedicated program, it features some filter facilities, and let the user browse the found duplications plotted on an interactive chord graph, that let the user select duplications to get more details on them, or directly download their sequences for further examination.

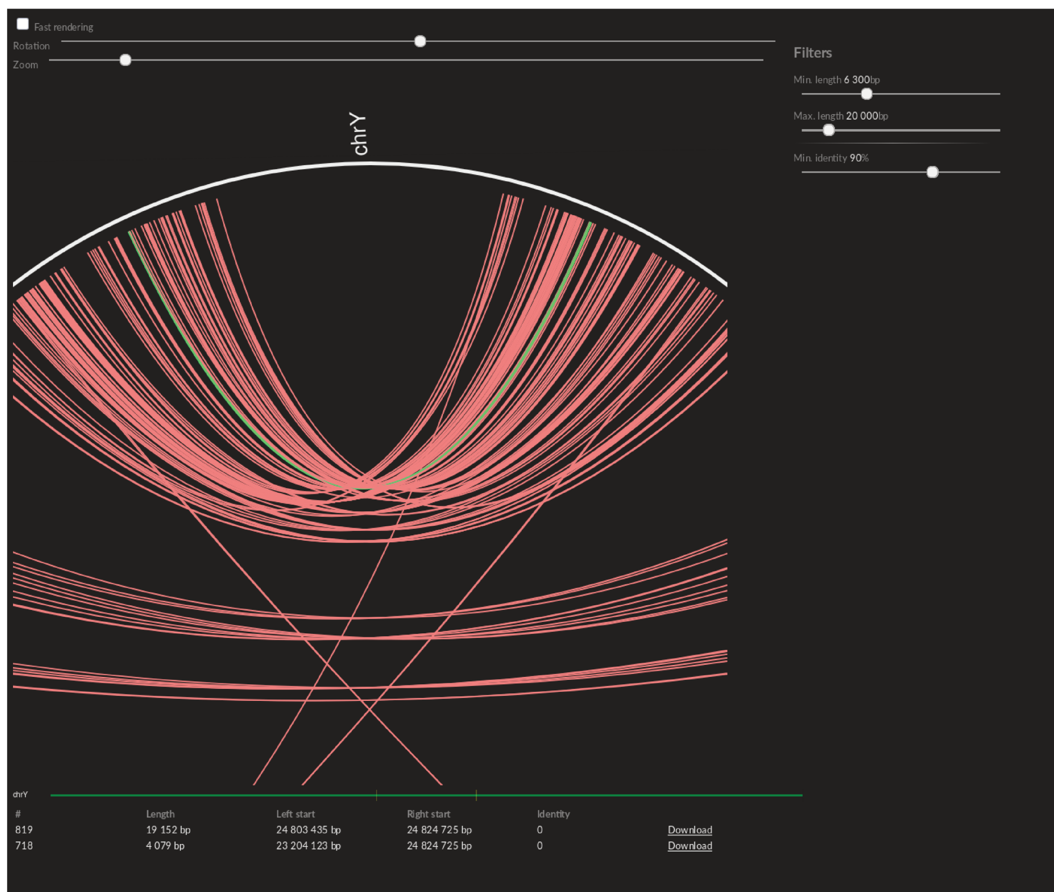


Figure 3.9: A glance at the web interface facilities for duplications visualization

Chapter 4

Results & Discussion

4.1 Benchmarking

As soon as the program was implemented, the first concern was to ensure its actual correctness on non-trivial data, and we used two methods in parallel to this end. The first one was to compare our result on actual genetic material to wet-lab studies on the same topic; and the second, to generate great quantities of artificial DNA seeded with segmental duplications, and measure ASGART performances compared both to the exactly-known list of made-up duplications on the one hand, and to the result of other comparable programs on the other hand, in precision, in running time and in memory use.

4.1.1 Artificial DNA

As a benchmark, we wanted to test ASGART precision compared to its main concurrents regarding the located duplications. To this end, we developed a simple DNA generator. This generator spawns artificial DNA fragment containing a variable number of non-interlaced duplications, each of them with their own length, number of duplicons, and degree of divergence between its duplicons (which is represented by varying identity rate), modeled by random alteration of the duplicons – be them insertions and deletions of various scopes, or simple point mutations. All of these parameters can, naturally, be set before generating the duplications-enriched fragment.

We settled on using artificial sequences ranging from 10,000 to 100,000bp (that allow for a limited run time when running numerous automated tests), containing duplications made of one to five duplicons, ranging from 1,000bp to 20,000bp, each of these duplicons exhibiting an identity rate from 90% to 99.99% with its brethren, these two last settings reflecting the bulk of the duplications found in real life data. We decided to compare ASGART to three tools representative of the current practices: a fast aligner, LAST[67]; a sequence searcher, MUMmer, combined with its post-processing script nucmer[29, 91], and the previously detailed Vmatch, all of them being used according to their manuals.

We generated 10,000 sequences, each of them containing perfectly known duplications, whose coordinates were stored apart for comparison with the results of ASGART, Vmatch, LAST and MUMmer. These sequences were then processed by the four mentioned programs, and the results saved for comparison with the original duplications. It should be noted here that MUMmer and LAST do not offer the possibility to expand multiple armed matches, *i.e.* high homology zones spawning several duplicons, and only extend two-armed ones. Therefore, we had to develop a few scripts taking as input results from LAST and MUMmer and gathering together matching zones of the same duplications family. The running time of these scripts were not counted in the final benchmarks. Finally, we plotted the results of this study in Figure 4.1.

The first interesting thing to note is that globally, all of the tested programs find more than 90% of the duplications with a shift of less than 20bp, which represents an error of only 2% for the shortest duplications (1,000bp), and a meager 0.02% for the longest ones (100,000bp). Second, with the exception of MUMmer exhibiting a whopping 35% of duplications found with less then 5bp of error (MUMmer/nucmer, unfortunately, hampered when processing larger fragments, as shown in Figure 4.9), ASGART globally follows the same distribution than these already established tools. Therefore, we can assert that ASGART satisfies our needs in precision when it comes down to pinpointing the position of the duplicons of a duplications family.

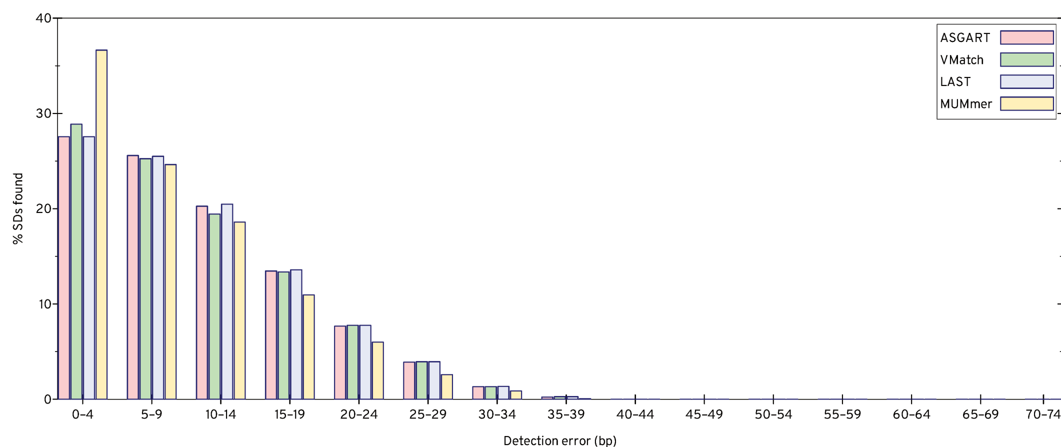


Figure 4.1: Histogram representation (truncated at 75bp) of the repartition of the error shift of duplications found by ASGART, Vmatch, LAST and MUMmer when ran on the procedurally generated reference duplications.

4.1.2 Natural DNA

Method

After a validation on artificial DNA, we established a benchmarking protocol to compare ASGART to the same set of tools on natural DNA. Due to performance issues that will be

detailed later, we excluded MUMmer/nucmer. It should be noted that LAST results would not be directly usable in a real-world case, as LAST only produces a list of two-pronged alignments and does not cluster them, thus masking the underlying families. As a source of reference SDs, we used the reference track available at the UCSC genome browser[134], curated from the study of the human SDs by Bailey *et al.*[13].

For each of these comparisons. The considered chromosome is split in 1,000bp long windows, that are then each tested for intersection with a duplicon of an SDs family from either benchmarked tool A and B. We define the *score* of A compared to B as $A/B = \frac{n_{A \cap B}}{n_B}$, and, symetrically, the score of B compared to A as $B/A = \frac{n_{A \cap B}}{n_A}$; where n_A is the number of windows intersecting with a duplicon as found by A, n_B the number of windows intersecting with a duplicon as found by B, and $n_{A \cap B}$ the number of windows intersecting with a duplicon as found by A *and* a duplicon as found by B. As the SDs are defined as being at least 1,000bp long, the use of 1,000bp-long window ensure a precise comparison of tools.

ASGART and Vmatch results were properly aligned to ensure the absence of false positives, LAST guarantees the absence of them[1], and the reference track is assumed to be curated.

Human Chromosomes

The Human Y Chromosome The human Y chromosome is a well studied subject, mostly due to its peculiar evolutionary pathway and its palindromes-rich structure, whose main duplicated areas range from a few thousands basepairs to over a million. Being a well documented source of natural data, it was deemed an excellent reference dataset for featuring a wide range of precisely mapped duplications varying in length, orientation, spacing, composition, etc. The human Y chromosome is relatively short, at *ca.* 50Mbp long. As a whole chromosome, it is both long enough to be biologically significant and short enough to be processed it in a negligible time (*ca.* 12 seconds of wall-clock time on a 5th generation i5, down to *ca.* 3 seconds when using the four cores), and so could be used as a dataset for automated testing.

When compared to the reference track, ASGART displays a score of 94.48%. Symetrically, the reference track exhibits a sensitivity of 76.96% when compared to ASGART results. It should be noted that the areas ASGART misses are mostly situated around large duplications clusters, which are detected nevertheless.

In comparison, Vmatch scores a score of 89.55% compared to the reference track, while the reference track itself displays a score of 78.02% with regards to Vmatch results. LAST does not go over a 73.82% score.

When compared to Vmatch, ASGART scores a score of 89.67% on Vmatch duplications, and Vmatch obtains a score of 82.97% on ASGART results. These low numbers should be taken in their context, as the large clusters of SDs are correctly detected by both Vmatch and ASGART. A majority of the divergence can be explained by the flawed determining of

[1] As it aligns all its results by definition.

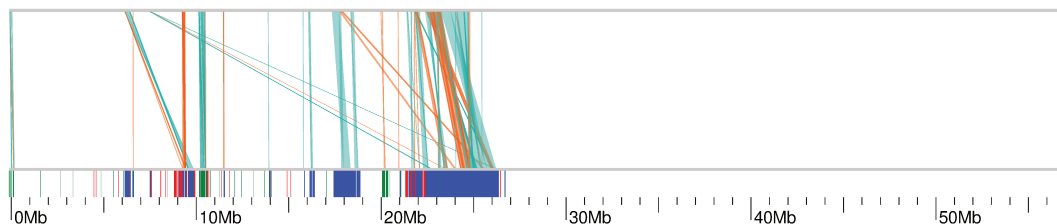


Figure 4.2: *Top:* the duplications found in the human Y chromosome by ASGART. The structures schematically represented in Figure 1.2 are easily identifiable, although at a higher resolution. *Bottom:* a comparison of the SDs found by ASGART in the Y chromosome compared to the reference track. Duplicated areas found in both the reference track and ASGART are shown in blue; duplicated areas only found by ASGART are in green; duplicated areas present in the reference track but not detected by ASGART are shown in red. (For practicality reasons, the graph is made with 10,000bp-long windows)

the very limits of duplication clusters, or of isolated, small-scale duplication by either tool. It should also be noted that ASGART exhibits better clustering abilities, as the SDs found by ASGART are gathered in *ca.* 5,000 families, whereas Vmatch SDs are scattered among more than 80,000 families.

The Human X Chromosome The X chromosome is another duplications-enriched fragment of the human genome – although not as well explored as the Y chromosome. Following the same protocol, when compared to the reference track (Figure 4.3), ASGART exhibits a score of 75.12%, to be compared with the equivalent score of 73.95% obtained by Vmatch and the far lower 51.06% scored by LAST. Once again, despite the low numeric values, most of the misses of ASGART and Vmatch actually lay on the fringe of clusters of duplications, tempering their impact in the context of real-life studies.

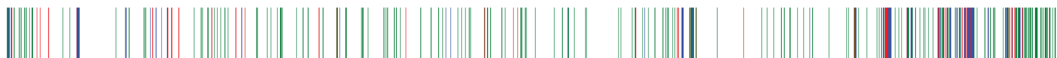


Figure 4.3: A comparison of the SDs found by ASGART in the human X chromosome compared to the ones in the reference track. Duplicated areas present both in ASGART results and the reference track are shown in blue; duplicated areas only found by ASGART are in green; duplicated areas only present in the reference track are shown in red. (For practicality reasons, the graph is made with 10,000bp-long windows)

However, ASGART also uncovers new SDs, as the reference track scores a score of only 25.52% on the SDs found by ASGART.

The Human Chromosome 11 We chose the chromosome 11 of the human genome as a typical autosome regarding its duplicated content. For this sequence, when compared to the reference track (Figure 4.4), ASGART scores a score of 76.09% on the reference track; for comparison, Vmatch obtains an equivalent score of 77.47% and LAST a lower one of 55.53%.

Similarly to what has been observed for the X and Y chromosomes, most of the mismatches between ASGART and the reference track are found near the limits of clusters of duplications. In opposition to previous results, a cluster is missed by ASGART in the subtelomeric region of the p arm, that ASGART skips due to its high percentage of undetermined nucleotides and its enrichment in localized, high-frequency, high duplicons count satellites.

Figure 4.4: A comparison of the SDs found by ASGART in the human chromosome 11 compared to the ones in the reference track. Duplicated areas present both in ASGART results and the reference track are shown in blue; duplicated areas only found by ASGART are in green; duplicated areas only present in the reference track are shown in red. (For practicality reasons, the graph is made with 10,000bp-long windows)

Again, ASGART detects quite a few new duplications clusters (particularly at the end of the q arm), as the reference track scores a score of only 34.19% on the SDs found by ASGART.

Summary & Discussion It emerges from this benchmark (in Table 4.1) that although the scores (Figure 4.5) are relatively low, ASGART exhibits scores comparable (Vmatch) or superior (LAST) to the existing programs. From a qualitative perspective, ASGART does not miss any structurally complex duplication cluster found either in the reference track or by Vmatch or LAST.

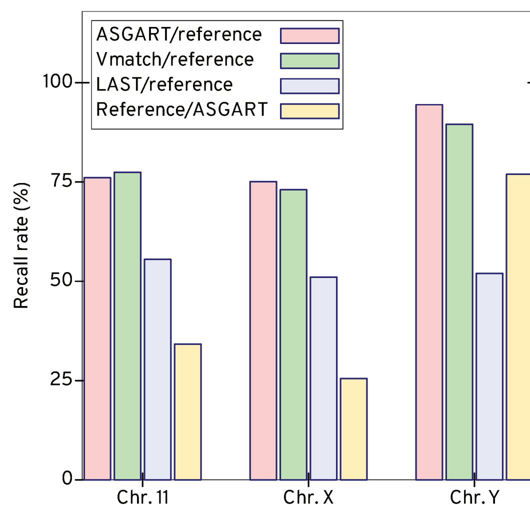


Figure 4.5: Scores of ASGART compared to the reference track, Vmatch compared to the reference track, LAST compared to the reference track, and the reference track compared to ASGART.

We have two main ways of explaining the mismatches between ASGART and the reference SDs track. First, ASGART and the reference track mostly differ on the fringe of duplication clusters (Figures 4.2, 4.3 & 4.4), where the identity rate start to plummet, as most

large mismatches between duplicons within an SD family are typically found on the extremity and in clusters inside the duplicons. Although contributing to lower ASGART score, this flaw is not crucial in real-world workflow, as a conservative preventive extension of a few thousands baisepairs of the duplicons pre-alignment is enough to mitigate its consequences.

Table 4.1: Scores of ASGART compared to the reference track, Vmatch compared to the reference track, LAST compared to the reference track, and the reference track compared to ASGART. Best scores are in bold.

	Chr. 11	Chr. X	Chr. Y
ASGART/reference	76.09%	**75.12%**	**94.48%**
Vmatch/reference	**77.47%**	73.05%	89.55%
LAST/reference	55.53%	51.06%	52.0%
Reference/ASGART	34.19%	25.52%	75.96%

Second, ASGART uses a definition of identity rate between duplicons different than the ones from the reference track: if ASGART counts indels as mismatches, the reference track ignores them when computing the identity rate between duplicons.

When compared to the state of the art, it appears that although ASGART and Vmatch results never coincide exactly, ASGART exhibits a satisfying score compared to Vmatch results, whereas the opposite is not necessarily true. Regardless of performance concerns, we can thus safely assumes that ASGART will allow its users to uncover new duplications clusters that would not have been found if using Vmatch.

Cross-Species Comparison

We then extended the test to a panel of chromosomes (Figure 4.6, Table 4.2) from several model organisms to gather data representative of larger scope studies. As we now want to compare these tools in an exploratory role, we compare ASGART performances to those of Vmatch and LAST (and *vice-versa*) rather than to established database. We selected the chromosome 22 of the zebrafish (*Danio rerio*), the chromosome 2 of arabidopsis (*Arabidopsis thaliana*), the previosuly studied chromosome Y of the human, and the chromosome 3 of the fruitfly (*Drosophila melanogaster*).

A noticeable result is the low score of ASGART when processing arabidopsis chromosome 2. After further inspection, it turns out that most of the missed duplications are mostly made of numerous satellite-type duplications, that are skipped on purpose by ASGART due to their very high number of identical k-mers. Although it strongly impacts the score, it is irrelevant in our context as we focus on detection of SDs made of more complex DNA material.

Despite this anomaly, ASGART displays scores equivalent or greater than the other tools when they are compared to ASGART, which hints toward ASGART generally detecting a higher number of SDs. This hypothesis is supported by the larger numbers of duplications

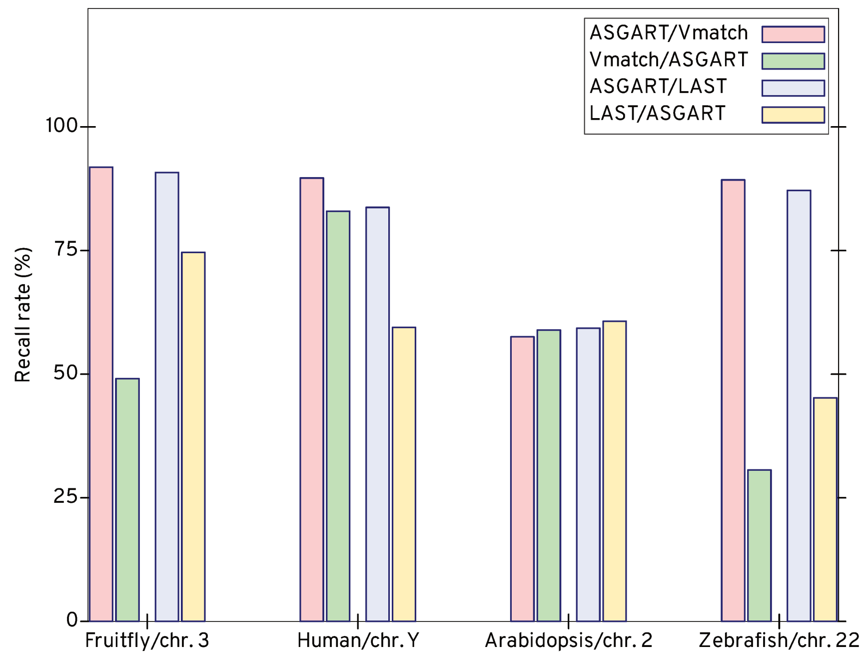


Figure 4.6: Scores of ASGART compared to Vmatch and LAST for a cross-species panel of test chromosomes.

Table 4.2: Score of ASGART compared to Vmatch and of ASGART compared to LAST on a panel of chromosomes from model organisms.

	Fruitfly, #3	Human, Y	Arabidopsis, #2	Zebrafish, #22
ASGART/Vmatch	91.88	89.68	57.56	89.30
Vmatch/ASGART	49.10	82.97	58.93	30.63
ASGART/LAST	90.79	83.73	59.30	87.17
LAST/ASGART	74.65	59.46	60.71	45.22

found by ASGART compared to Vmatch (Figure 4.7) and LAST (Figure 4.8).

Moreover, if it is not reflected in these numbers, ASGART is more efficient in finding longer and better clustered duplications families. During our tests, we typically observe a number of families one or two order of magnitudes larger for Vmatch or LAST compared to ASGART. A better linking of longer duplicons is an asset for genomic studies, as it gives a better overview of both the structure and the evolutionary path of the concerned fragments, and thus concentrate more information than a more scattered interpretation of the same underlying genetic material.

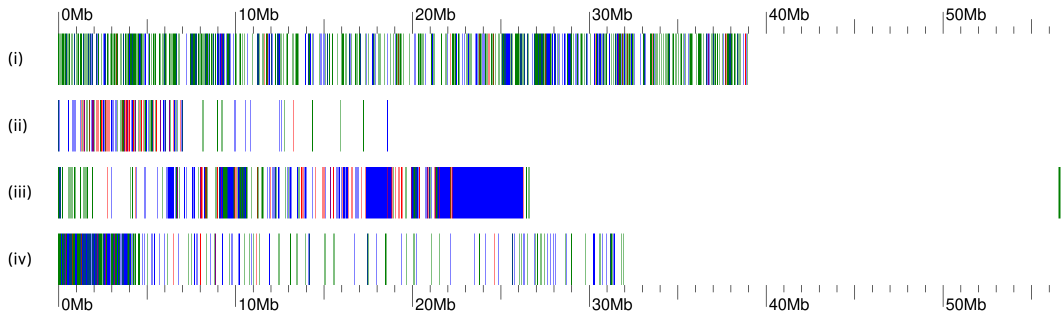


Figure 4.7: Comparison of the duplications found by ASGART and Vmatch in (i) the chromosome 22 of the zebrafish (*Danio rerio*), (ii) the chromosome 2 of arabidopsis (*Arabidopsis thaliana*), (iii) the chromosome Y of the human; (iv) the chromosome 3 of the fruitfly (*Drosophila melanogaster*). Duplications found by both ASGART and Vmatch are shown in blue; duplications found by ASGART only are shown in green; duplications found by Vmatch only are shown in red. (For practicality reasons, the graph is made with 10,000bp-long windows)

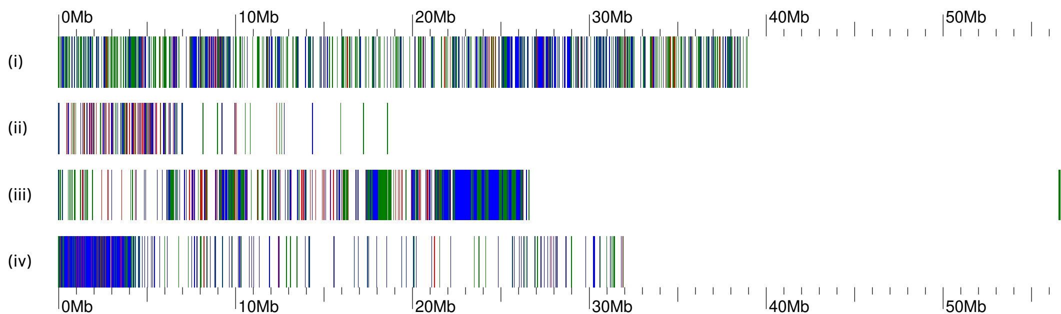


Figure 4.8: Comparison of the duplications found by ASGART and LAST in (i) the chromosome 22 of the zebrafish (*Danio rerio*), (ii) the chromosome 2 of arabidopsis (*Arabidopsis thaliana*), (iii) the chromosome Y of the human; (iv) the chromosome 3 of the fruitfly (*Drosophila melanogaster*). Duplications found by both ASGART and LAST are shown in blue; duplications found by ASGART only are shown in green; duplications found by LAST only are shown in red. (For practicality reasons, the graph is made with 10,000bp-long windows)

Similarly to what has been observed in the previous benchmark, ASGART and Vmatch both agree on large SDs clusters, and only slighty diverge on their fringe (Vmatch *e.g.* better

extending some palindrome of the human Y chromosome). The main source of difference is found in isolated SDs: although both programs typically share a common set of results, ASGART tend to be more sensible to small-scale SDs than Vmatch (*e.g.* in the chromosome 22 of the zebrafish or the chromosome 2 of the fruitfly). But from a large-scale, structural, perspective, both tools exhibit similar results.

The differences between ASGART and LAST results on the same panel of chromosomes can mostly be attributed to the functional divergence of the two programs. Outside of performance concerns, ASGART exhibits better results than LAST in large duplications clusters (typically, the human Y chromosome), whereas LAST shines in the areas enriched in high-frequency, low-signal repetitions (*e.g.* the chromosome 2 of *A. Thaliana*).

4.1.3 Performances Benchmarking

The next step was to compare ASGART performances to its competitors on a larger scale. To this end, we selected a panel of chromosomes within several genomes, discriminated on three criteria.

Firstly, we settled on using sequencing data from so-called reference genomes, so that the sequenced data had the best chances to be of high quality, which is of a great importance, especially when it comes down to duplication detection, for the reasons detailed earlier. Moreover, they are the most well studied genomes, hence have their duplications covered best in existing literature, which gives us a solid base for further comparisons.

Secondly, we chose to take chromosomes whose sizes would cover the spectrum of chromosome sizes of the considered species, so a "large" chromosome, a "medium" chromosome, and a "small" chromosome were chosen. Thus, we could get a general idea of resources consumption scaling regarding chromosomes sizes depending on the species.

Thirdly, whenever possible, we added an haploid sexual chromosome. Due to the difficulty of sequencing such chromosomes and the fact that not every reference genome have them (as species may use other mechanisms for sex determination), they were naturally not always available. But having haploid chromosomes in our panel allows us to get a first idea of the variation in duplicated content between autosomes and sex chromosomes.

Eventually, we selected:

- chromosomes 1, 10, 21 & Y for the human (*homo sapiens*);
- chromosomes 1, 7, 19 & Y for the mouse (*mus musculus*);
- chromosomes 4, 17 & 22 for the zebrafish (*danio rerio*);
- chromosomes 3, 2, 4 & Y for *drosophila melanogaster*;
- chromosomes 1, 2 & 4 for *arabidopsis thaliana*.

Table 4.3: Performance comparison of CPU time and memory usage of ASGART, LAST and MUMmer on chromosomes of various sizes and from different species – OoM means Out of Memory, italic values are the best of their category. It should be noted that the indicated CPU times are user time; thanks to its parallelization, ASGART can decrease its wall-clock time with the number of available cores.

Species	Chr	Size(Mbp)	CPU Time (s)				Memory (GB)			
			ASGART	Vmatch	LAST	MUMmer	ASGART	Vmatch	LAST	MUMmer
H. Sapiens	1	241	*2,380*	8,471	12,996	OoM	*2.9*	3.1	9.8	OoM
	10	130	*953*	4,662	6,320	OoM	*1.6*	*1.6*	4.9	OoM
	21	46	*203*	1,302	1,528	67778	*0.6*	0.8	1.3	12
	Y	56	*32*	716	1,101	14205	*0.6*	0.9	1.1	3.6
M. Musculus	1	190	*2,728*	9,878	6,920	OoM	*2.3*	2.4	3.4	OoM
	7	141	*1,089*	6,440	4,807	OoM	2.3	*1.8*	3.4	OoM
	19	60	*146*	2,225	1,087	OoM	*0.8*	*0.8*	1	OoM
	Y	89	*604*		4,932	OoM	*1.1*	1.8	1.8	OoM
D. Rerio	4	76	*611*	13,151	2,050	OoM	*0.9*	1	3.7	OoM
	17	53	*738*	2,171	1,630	OoM	0.7	*0.6*	3.3	OoM
	22	39	*694*	1,740	760	OoM	*0.5*	*0.5*	1.9	OoM
D. Melanogaster	3	32	*75*	2,220	1,113	423	*0.37*	0.39	1.9	4.9
	2	25	*34*	1,382	489	115	0.32	*0.28*	0.4	9.5
	4	1.4	*0.54*	71	19	6.4	0.04	*0.02*	0.9	1.3
	Y	3.6	*2.6*	469	195	23.4	*0.06*	0.07	0.2	3.5
A. Thaliana	1	30	*22.5*	1,324	163	2173	0.41	*0.36*	0.5	*0.3*
	2	20	*12.7*	880	76	317	0.27	*0.23*	0.3	0.3
	4	19	*16.5*	771	90	622	0.25	*0.22*	0.3	0.25

This panel established, we screened each of these chromosomes with ASGART, Vmatch, LAST (being a representative aligner used as a matches source for other duplication finders, and thus present a minimal constraint on their total resource consumptions) and MUMmer/nucmer. All these tests were run on the same computer, and we recorded user time instead of wall-clock time consumption. Thus, it should be noted that ASGART wall-clock time consumption on four cores was roughly a quarter of the recorded user time. The results of this study are available in Table 4.3, and feature a few noticeable points.

First, and most important for us, ASGART is the winner – sometimes close – in nearly all of these tests when it comes down to resource consumption. But if being the fastest is a welcome advantage, what is more interesting is consuming less memory. Indeed, memory consumption is often the main problem when processing large datasets. If CPU consumption problems can be "solved" – for liberal interpretation of solved – by waiting longer for a result, memory resources are hard-capped; so the lesser a program consume, the better it will be able to cope with larger data. And it is obvious that nucmer, falling short of memory on a 32GB system for single chromosomes, is unfit for whole-genome studies. Similarly, Vmatch and LAST prove to be far slower that ASGART in worst cases, with ratio going over 10 in favor of ASGART on most extreme test cases. This does not take into account ASGART multithreading capacities, that can reduce wall-clock time, on the one hand, and the running time of the duplicon families gathering scripts needed to post-process raw LAST results in a usable form on the second hand.

Secondly, the variety in the content of duplications between all these screened chromosomes is reflected by the apparent lack of simple relationship between the size of the chromosome and the resources used, regarding both CPU time (Figure 4.9) and memory (Figure 4.10). Thus, whatever the tool, it is not really possible to estimate the time needed to screen a chromosome *before* screening it, which may complicate the case of users running exploratory screening in resource-limited environments.

4.1.4 Parallel Scaling

Besides single-core performances, efficient scaling over the number of available core was a main goal of ASGART to ease scaling over large datasets. ASGART implements parallelization by slicing the first input strand in chunks of a fixed size. These slices are then concurrently and individually processed as so many single-thread jobs by ASGART. The partial results are then merged, in a classical application of the map and fold strategy. This approach allows a parallelism implementation with few synchronization steps and close to no writable shared memory, making it safe and easy to implement.

However, as the slicing is spatial (over the length of the inputs) and not temporal (over the total computation time), an optimal use of the available threads for the whole duration of the computation can not be ensured. Indeed, all the chunks will not take to same time to be processed. This discrepancy may lead to a few threads processing the more complex areas

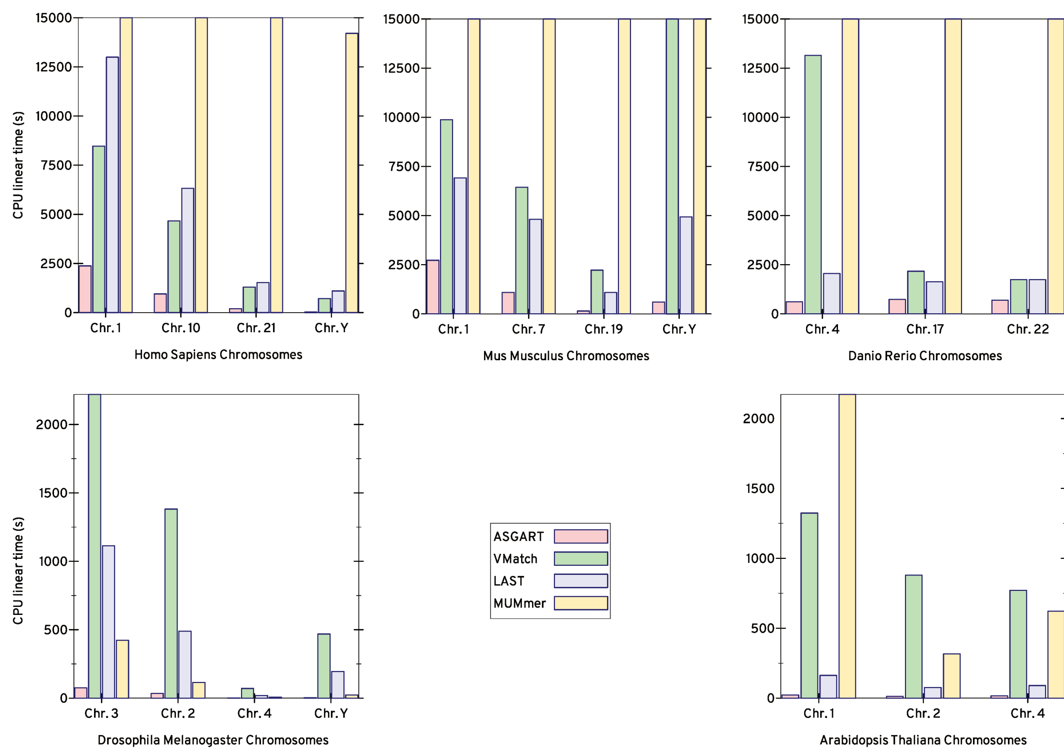


Figure 4.9: Representation of the user CPU time on one core for ASGART, Vmatch, LAST and MUMmer when screening the chromosome selected in the benchmark. The bars reaching the top of the frame symbolize an out of memory error.

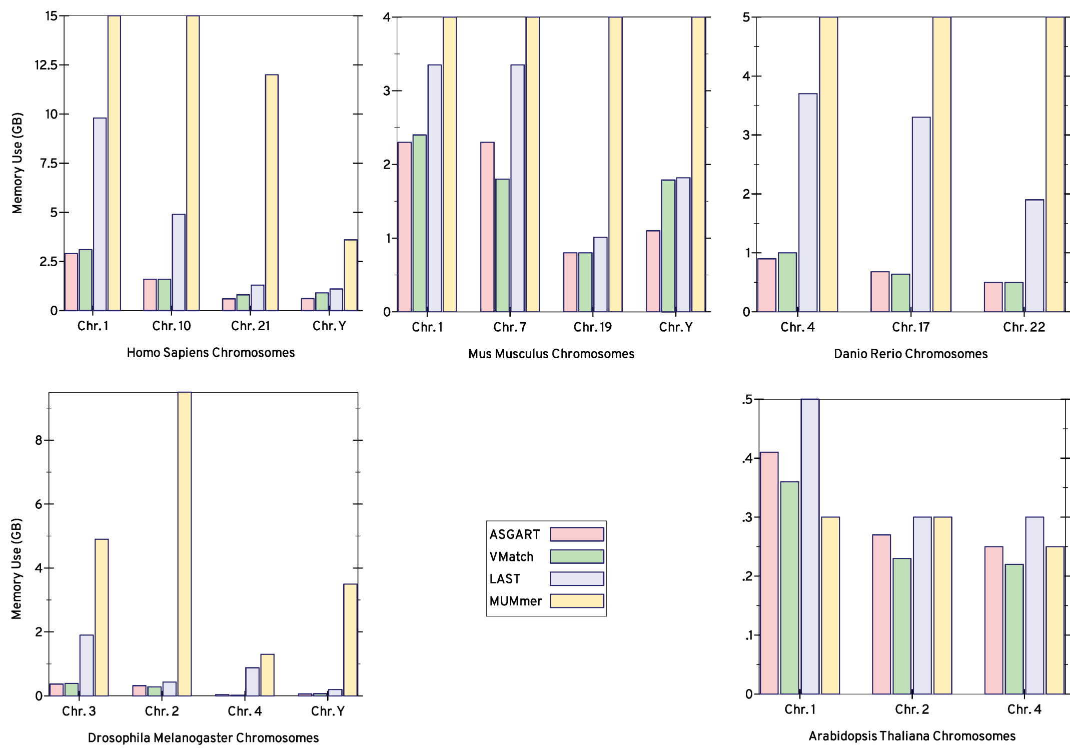


Figure 4.10: Representation of the peak memory use of ASGART, Vmatch, LAST and MUMmer when screening the chromosome selected in the benchmark. The bars reaching the top of the frame symbolize an out of memory error.

finishing far later than the others, without a possibility to split further the remaining work.

For an illustration of this limitation, we benchmarked the processing of the chromosome 1, the longest of the human genome at 250Mbp, with an increasing number of threads. The time consumption profile is shown Figure 4.11, and the speedup factor Figure 4.12.

Incidentally, a drastic fall of efficiency is observed when starting to use Intel's *Hyper-Threading* virtual cores instead of more physical cores. When hitting this limit, the saturation of the CPU will lead to time being just wasted in context switches with a notable cost in user-time without improvement to the wall-clock time. This phenomenon highlights the CPU-boundedness nature of ASGART.

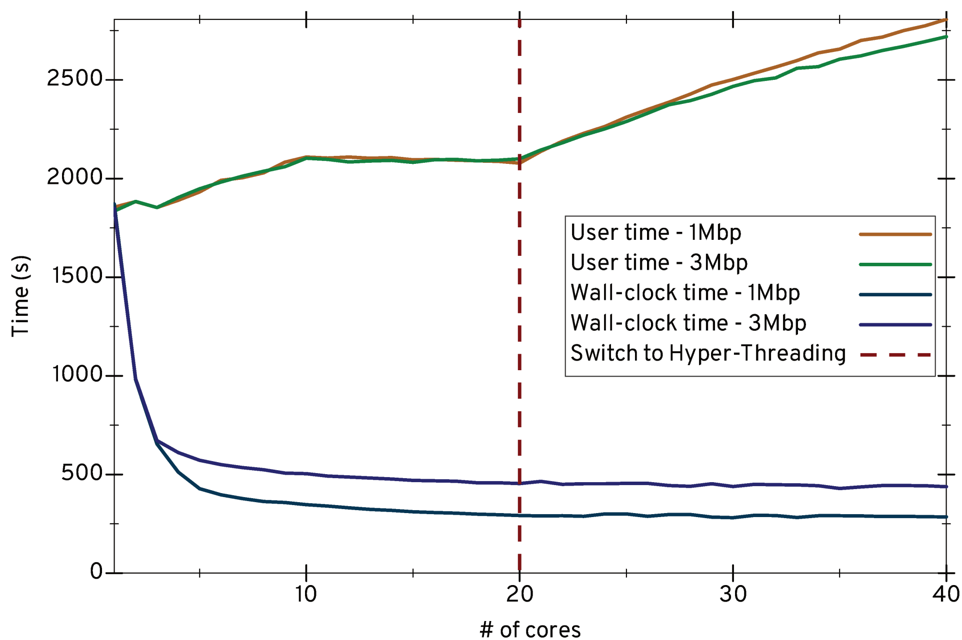


Figure 4.11: ASGART parallelization effect on the processing of the human chromosome 1 (~250Mbp) using 1Mbp chunks and 3Mbp chunks.

When considering scaling over physical cores, the difference in slice size does not have an impact on the global performances, as the user time consumption profile is virtually identical in both cases. However, switching from larger (3Mbp long) to smaller (1Mbp long) chunks has a large influence on the maximal speedup reached: in the same conditions, 3Mbp chunks result in a maximal speedup of ×4, while 1Mbp chunks let ASGART reaches a speedup of ×7.

The underlying reasons for this discrepancy can be illustrated with the comparison CPU use over time (Figure 4.13). If both slicing sizes lead to a similar amount of time spent with all available threads used, the processing of the last chunks, when there remain less of them than available threads is the main cause of ineffeciency in parallelization scaling.

For example, when using 20 threads (on physical cores), ASGART spends *ca.* 130" using all threads, indifferently from using 1Mbp or 3Mbp chunks. However, the processing of the remaining data from this point differs in the two cases. If the 1Mbp slicing makes ASGART

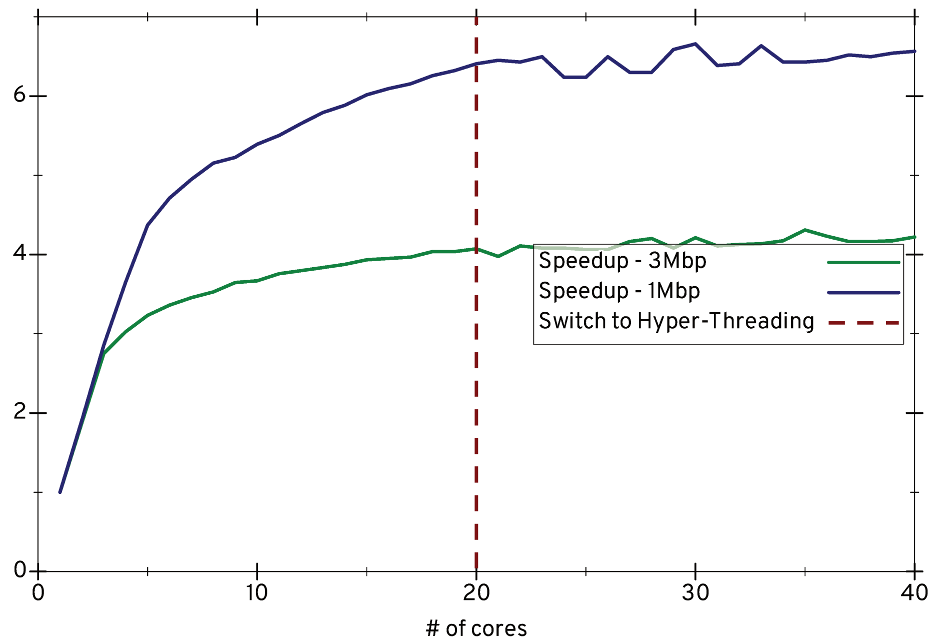


Figure 4.12: Speedup of ASGART regarding the number of cores used while processing the human chromosome 1 (~250Mbp) using 1Mbp chunks and 3Mbp chunks.

finish the work with 3 threads for ∼50”, 2 threads for ∼70” and one thread for only ∼30” (for a total of ∼150”-long non optimal use of the CPU), the 3Mbp slicing is far less efficient. It makes ASGART work ∼50” on two threads, then ∼250” with a single thread processing the obviously complex remaining chunk (for a total of ∼300”-long non-optimal use of the CPU). The finer slicing obviously allows for a better repartition of the hard to process areas of the input sequences among more threads.

This sensitivity to the slicing size explains the discrepancy in the speedup efficiency: if it does not noticeably influence the intrinsic efficiency of ASGART (the total user time is not affected), it has a strong effect on the wall-clock time. ASGART uses a default chunk size of 1Mbp, but we let the users customize this setting, in order to let them use a sensible chunk size according to their use case. It should be large enough to avoid to lengthen too much the folding of the multiple partial results, but short enough to alleviate the impact of hard to process areas of the input sequences on the final wall-clock time.

4.2 Whole Genomes Benchmarking

Once an acceptable precision was ensured, the next step for us was to run ASGART on large-scale, whole genome analyses, to confirm both its satisfactory performances and its biological correctness; and, eventually, to compare its results with the existing literature concerning whole-genome duplications analysis.

To this end, we simply launched mapping jobs on the whole genome of the five previ-

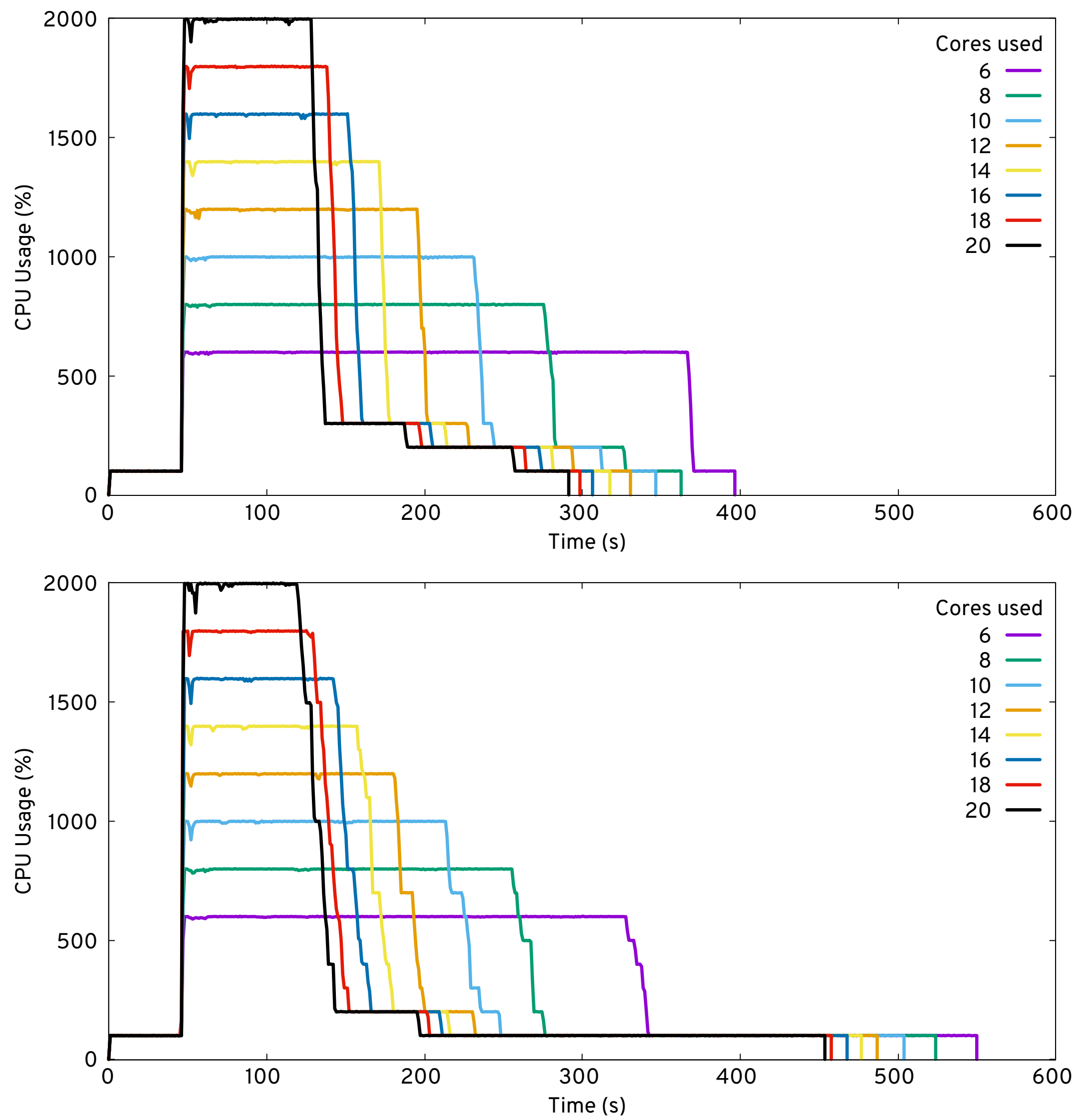


Figure 4.13: Running profile of ASGART when using 1Mbp long (top) and 3Mbp long (bottom) chunks. The first plateau is the single-threaded preprocessing step.

ously mentioned reference species (instead of a selected panel of chromosomes) while keeping track of the consumed resources. The repetitions-rich and coding DNA-poor telomeric and pericentromeric regions were not masked (as it is usually done), to keep the study as exhaustive as may be. The process was run on Eos, CALMIP's supercomputer[20]. Eos is a cluster of compute nodes, each of the 612 of them being equipped with two 10-cores Intel Xeon E5-2680@2.8GHz and 64GB of RAM. However, due to limitations in the queuing mechanisms, managed by Slurm, we could only use 20 of these nodes at once. Parallelization was done on two levels. On a first level, the twenty available physical cores were used on every node. We voluntarily limited ourselves to twenty threads matching the twenty physical cores instead of the forty logical cores for a simple reason: our use of Eos was billed according to user time consumption, and the acceleration resulting from the use of logical cores was, albeit noticeable, more limited than the one coming from the use of physical cores. Therefore, it made sense to prefer an increase in wall-clock time rather than in time billed.

On a second level, the input genomes were cut in twenty equally long parts (as many as simultaneously available nodes), each of them processed on a different compute node, the resulting twenty partial results being merged afterward. All these segmentations were slightly overhanging, so that no duplications family that would have fell right on a seam would be lost. The results of this study are synthesized in Table 4.4.

As already observed when working on single chromosomes, the time spent processing each of these genomes does not exhibit any coherent relationship between the size of the genome in base pairs and the user CPU time used. That discrepancy was expected (see 3.2.4), as it is already known that if all known genomes contain large quantities of duplications, the proportion of them relatively to the total size of the genome vary largely from one species to the other, as well as their characteristics. The zebrafish is well-known for its repetition- and duplications-rich genome and it shows in our result, as it required nearly as much CPU time as the human genome, for a genome length less than half as short.

Another remark is the relatively low amount of memory used compared to single chromosome processing: whereas ASGART needs 2.8GB to process the single first chromosome of the human genome, we only needed 9GB to process the whole human genome. The trick here is that instead of comparing, at once on a single machine, the whole human genome to itself, the inputs were split. Indeed, we split each machine job in a few dozen subjobs by dividing the human genome in slightly overlapping 100Mbp chunks, against which the human genome was successively compared. The results were then concatenated in a single one, equivalent to the comparison of the whole genome against itself. An important detail here is that it is cheaper in memory to compare the human genome to a chunk of it rather than the opposite, as ASGART builds a suffix array of the second fragment; and the suffix array size is directly proportional to the size of the underlying string. Thus, combining these two tricks allows us to keep a comparatively low memory usage. In any case, the RAM consumption follows a nearly linear scaling (Figure 4.14). This was expected, as the bulk of RAM that

Table 4.4: Characteristics of segmental duplications mapped in the 5 model organisms studied.

	H. Sapiens	**M. Musculus**	**D. Rerio**	**D. Melanogaster**	**A. Thaliana**
Analyzed genome size	2.9GB	2.7GB	1.4GB	134MB	119MB
Chromosomes (2N)	23	20	25	4	5
User CPU time	371h	263h	311h	241s	481s
Peak memory usage	9GB	8GB	4GB	1.7GB	1.4GB
Intra-chr (mean(stdev))	5.69(2.92)%	10.13(19.5)%	4.75(5.72)%	6.81(2.19)%	1.97(0.79)%
Inter-chr (mean(stdev))	4.09(2.75)%	5.39(1.13)%	7.36(1.79)%	4.98(1.13)%	1.78(0.13)%
SDs (all, in Mbp)	187	265	115.8	5.4	2.98
SDs (>20 Kbp, in Mbp)	70.8	70.5	2.78	0.28	0.29
SDs (%genome)	6.5	9.8	8.64	6.32	2.5

ASGART consumes is used to store the input strands and their suffix array, that are obviously linearly increasing with the length of the input strands.

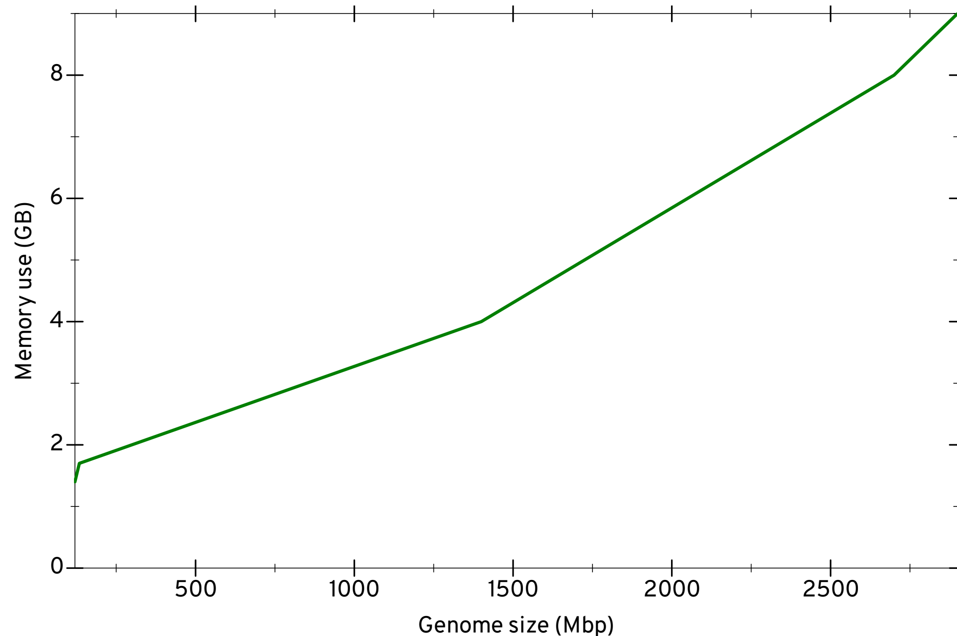


Figure 4.14: ASGART memory consumption compared to the processed genome length.

Globally, SDs represent less than 10% of every genome surveyed, with a minimum of ~2.5% observed in *Arabidopsis thaliana*, and a maximum of ~8.6% in *Mus musculus*. Concerning the human genome, this number is close to what is mentioned in the previous estimations[85]. Indeed, if this study mentions a lower rate of duplications in the human genome, it also mentions the limitations of the methods used then and conjectures an actual duplication rate closer to 6 to 7%. Our result of a global duplication rate a bit over 6.5% is coherent with this forecasting.

It is interesting to note that for all studied genomes with the exception of the zebrafish, intra-chromosomal duplications occur at a higher rate than inter-chromosomal duplications. This is coherent with SDs generally arising from non-allelic homologous recombination, that, when successful, takes place in an overwhelming majority of cases on a single chromosome.

Large SDs tend to be uncommon compared to smaller ones, with the relatively lower content of large SDs being owned by, once again, the zebrafish. It is, again, coherent with NAHR generally occurring on small part of the chromosome, duplications thus growing rarer when their size increases.

To summarize, this small study results are twofold. First, they confirm that ASGART is able to find to an excellent degree of precisions the duplications already detected in wet lab and published in the literature, both in position and in characteristics. Secondly, these results can now be obtained with cheap hardware requirement. A few hundred hours of user

CPU time – resulting in a far lower wall-clock time thanks to ASGART performances scaling efficiently with the number of available cores – and a few gigabytes of memory are enough for any user to process a genome in its entirety, be it a trivial one like the drosophila or an especially duplications-rich one like the zebrafish.

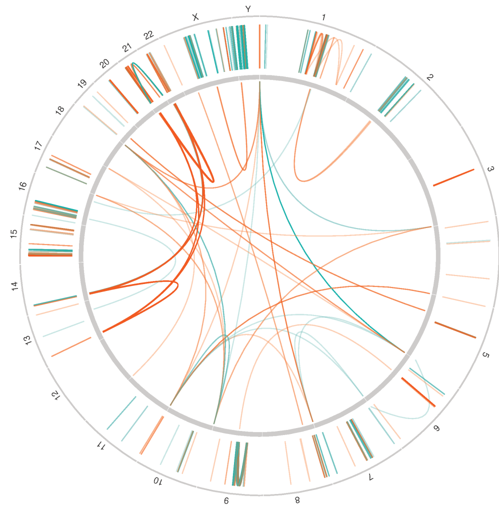


Figure 4.15: The SDs found by ASGART in *H. Sapiens* genome; direct in orange and palindromic in teal. For clarity sake, only the ones longer than 10,000bp are shown.

4.3 Application

4.3.1 Duplications in Sex Chromosomes Evolution

Now that ASGART can be considered stable, we are using it to explore the role played by duplications in genomes evolution, especially concerning sex chromosomes. Sex chromosomes tend to be enriched in duplications, and this misleads sequencing efforts; typically, the content of known duplications in sex chromosomes strongly increased after the adoption of *ad-hoc* methods to help with this ampliconic content. This has already been noticed during the subsequent sequencing of the human, mouse and chimpanzee genomes. We now intend to lead a study on the quantity and content of duplications in the sequenced genomes including sex chromosomes, as available either in the Ensembl or the NCBI databases.

To explain the peculiar stability of the human Y chromosome, and, especially, of the MSY region, the main hypothesis boils down to gene conversion through NAHR between its highly-similar duplicons, especially the large palindromic structures. Indeed, these structures can be altered in two main ways: either duplications or deletions, or by point mutations. It

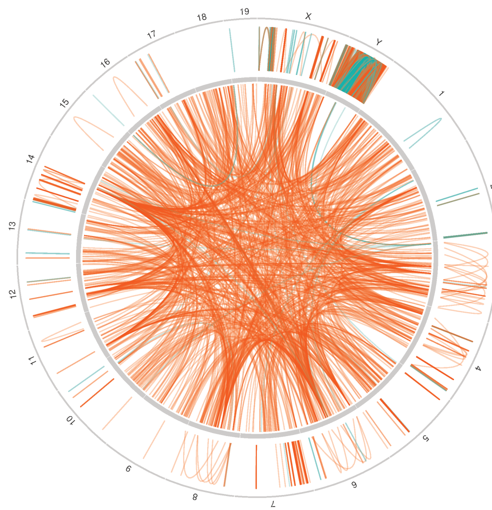


Figure 4.16: The SDs found by ASGART in *M. Musculus* genome; direct in orange and palindromic in teal. For clarity sake, only the ones longer than 10,000bp are shown.

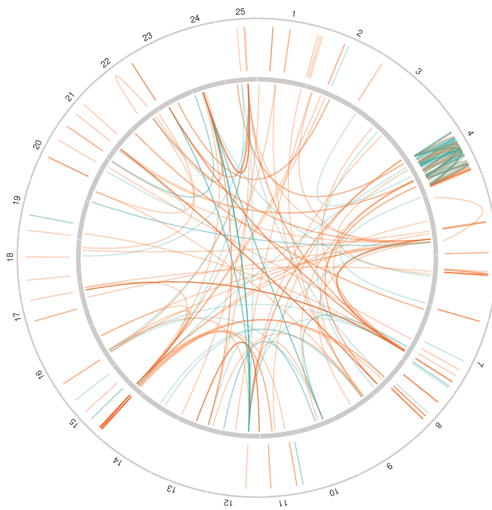


Figure 4.17: The SDs found by ASGART in *D. Rerio* genome; direct in orange and palindromic in teal. For clarity sake, only the ones longer than 10,000bp are shown.

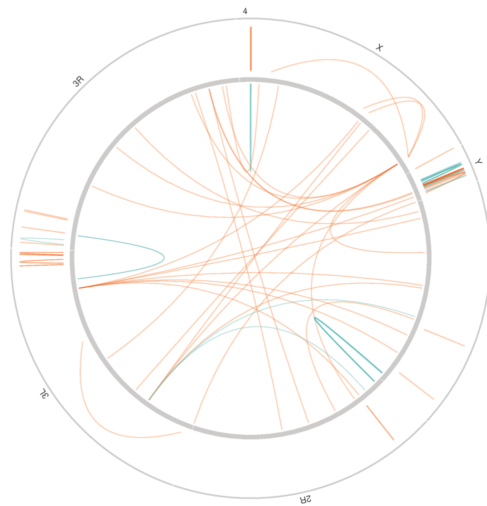


Figure 4.18: The SDs found by ASGART in *D. Melanogaster* genome; direct in orange and palindromic in teal. For clarity sake, only the ones longer than 10,000bp are shown.

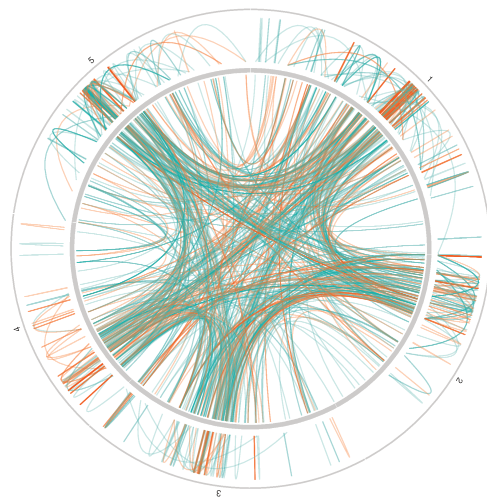


Figure 4.19: The SDs found by ASGART in *A. Thaliana* genome; direct in orange and palindromic in teal.

is established that gene conversion between duplicons can rescue these altered structures by reinstating them to their previous state, using other duplicons as a template. And as most of the alterations of these structures lead to male sterility, the one escaping the repairing sieve of gene conversion end up purged by selection. This results in a glaring uniformity of MSY structure in men[14, 127].

Concerning the advantages imparted by this specificity, two parallel explanations may be proposed. The first one calls upon the genes regulation system, and explains the stability of the duplicons count by the need to keep a correct dosage of the genes they encompass. The linkage of dosage imbalance of some of these genes with troubles such as male fertility troubles or autism are clues on which this idea lays.

Another benefit of the highly similar duplicons scattered on the Y chromosome is the ompensation of the meiotic inactivation. As previously mentioned, the Y chromosome can only recombine with its paired chromosome – the X chromosome – on the comparatively tiny PARs. But, by exhibiting such alluring substrates for NAHR such as the palindromes arms and other repeated structures, the Y chromosome can, in some way, recombine with itself, thus generating variation in its own genetic material. Without this mechanism, the Y chromosome would be hard-pressed to find a way to create variation in the MSY region.

There is currently no indication to whether these mechanisms are human-only, shared with the mammalian clade, or even with other species using similar sex-determination systems, such as the Z/W system. Therefore, we set on using ASGART to map palindromic duplications in other species, mammals or not, whose sex chromosomes have been sequenced. We want to statistically compare the duplications content rated between the autosomes and the sex chromosomes. The final goal would be to determine whether the mechanisms already known for the human Y chromosome might be extrapolated to sex chromosomes of other mammals, or even to other species featuring a chromosome-based sex-determination system, such as species exhibiting the Z/W chromosomes.

To this end, we ran ASGART on all of the sequenced genomes containing sex chromosomes available either through the NCBI or Ensembl databases (*bos indicus*, 1.0; *cynoglossus semilaevis*, Cse v1.0; *chlorocebus sabaeus*, 1.1; *drosophila melanogaster*, BDGP6; *gallus gallus*, GRCg6a; *macaca mulatta*, Mmul 8.0.1; *pan troglodytes*, Clint PTRv2; *rattus norvegicus*, Rnor 6.0 and *sus scrofa*, Sscrofa 11.1). This dataset amounted to ten species having both their X/Z and Y/W sex chromosomes sequenced, and 25 species having only the X/Z one sequenced (Table 4.5).

Table 4.5: Repartition of the species with sequenced sex chromosome we mapped.

Sex chr. available	Species count	Mammals	Birds	Fishes	Insects
Both X&Y or Z&W	10	5	3	1	1
Only X or Z	25	20	4	0	1

We plotted the distribution of the palindromic duplications content found by ASGART for the autosomes of each species using the standard box plot representation, on top of which we superimposed two points, for the X/Z and Y/W sex chromosomes (Figure 4.20).

If in an overwhelming number of cases, the sex chromosomes score high, or the highest, in relative palindromic content, *drosophila melanogaster* is the only case where one of them is scoring very low, under the autosome average. We are not sure why this species is alone in exhibiting this behavior, but outside of the simplest explanation that the X chromosome of the drosophila has indeed a low palindrome content, we have two other hypothesis. First, this peculiarity may be the consequence of the large usage of the drosophila as a widely bred model organism, and the subsequent highly inbred nature of the lab individuals, including the ones used for the genome sequencing and resulting in a drifted genome compared to the outbred lineages. Or it may be the direct consequence of the difficulty of sequencing duplicated sequences, that would result in an underrepresentation of the duplicons content of this chromosome.

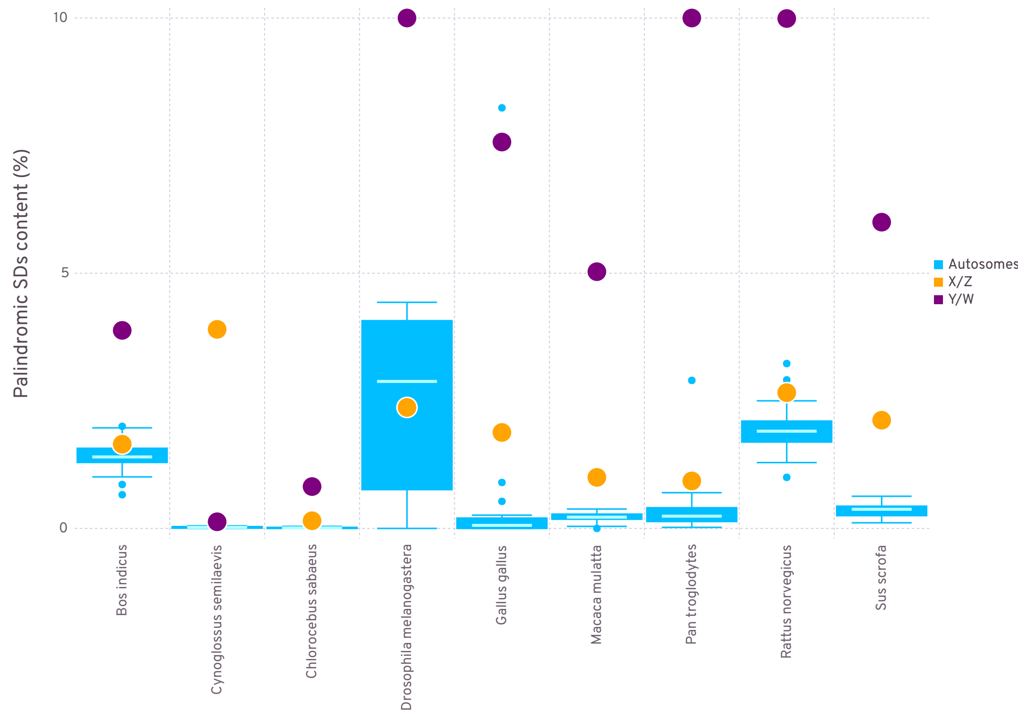


Figure 4.20: Boxplot of the palindromic content ratio observed in the species for which both of the sex chromosomes were sequenced. *Drosophila melanogaster* and *Pan troglodytes* Y chromosomes content were artificially capped at 10% for the sake of readability; they actually contain respectively 25.88% and 40.61% of palindromic content according to ASGART results.

These results show that, like in the human case, sex chromosomes of these other species are typically enriched in palindromic duplications when compared to their autosomes. The

door is now open to a more thorough exploration of this dataset.

4.3.2 Ongoing Work: Fertility Genes & SDs

From these preliminary results, we are exploring several leads. First, the duplications of the human Y chromosome contain many crucial genes, linked *e.g.* with male fertility and cerebral development. If the currently accepted hypothesis is that this arrangement ensures the integrity and the conservation of these genes over time thank to gene conversion, we would like to explore the gene content of palindromic duplications for the other sex chromosomes to determine what kind of gene sets they encompass, and if they are comparable to the human one. A strong correlation would be another clue hinting toward the confirmation of palindromes as structures ensuring the conservation of important genes in male lineages.

Second, the sequencing of the sex chromosomes, especially when they have a high content in duplicated sequences, is known to be complex and obtaining precise results is arduous. Therefore, we would like to explore more deeply the data contained in the so-called unplaced sequences and unlocalized sequences available for most of the sequenced species. These fragments are typically highly duplicated, and exploring the relations between their duplicated sequences and the one found in sex chromosomes could prove interesting. We chose to explore more in depth the results concerning *sus scrofa*, chosen at random among the species whose both sex chromosomes have been sequenced. In this peculiar case (Figure 4.21), some interesting points can be listed.

First, concerning the link between the Y chromosome, unlocalized sequence and unplaced sequence sequences, a first point is that, for *sus scrofa*, the entirety of unlocalized sequence sequences are belonging to the Y chromosome. And, as can be seen on the the figure, a consistent fraction of these fragments map to highly duplicated fragments of the Y chromosomes, mostly around the 7Mbp, 20Mbp and 40Mbp marks. This situation reflects the difficulty in assembling sex chromosomes and their highly duplicated content. Similarly to what was discovered through the subsequent sequencing and assembling of the human and mouse genomes, it suggests a global underestimation of the quantity of duplicated Y-linked sequences, and we expect to see the count of duplications discovered in newly sequenced or assembled genomes to strongly increase with the improvement of sequencing technologies. When it comes down to the unplaced sequences, although they may come from any chromosome in the genome, some of them exhibit large number of similarity among all of the Y chromosome, with no striking predominance in the highly-duplicated areas – contrary to the unlocalized sequence fragments.

Second, as is the case for the human Y chromosome, large palindromic areas tend to strongly correlate with the position of coding genes (especially around the 10Mbp and 40Mbp marks). This observation strengthens the hypothesis stipulating that palindromes may tend to be, as in the human Y chromosome, strong, evolutionary stable structures that preserve the integrity of their genetic payload. Further similar comparisons on the other species should

help gather additional data regarding this role. A correlation between palindromic duplications and fertility genes, as can be observed in the human, would be a great indicator of the same mechanisms happening in multiple species.

Third, some relatively large duplicons, both direct and palindromic, between the unplaced sequence and the Y chromosome can be observed in the 6Mbp-11Mbp window and should be studied further, to determine *e.g.* their composition and their potential connection to the gene content of the chromosome.

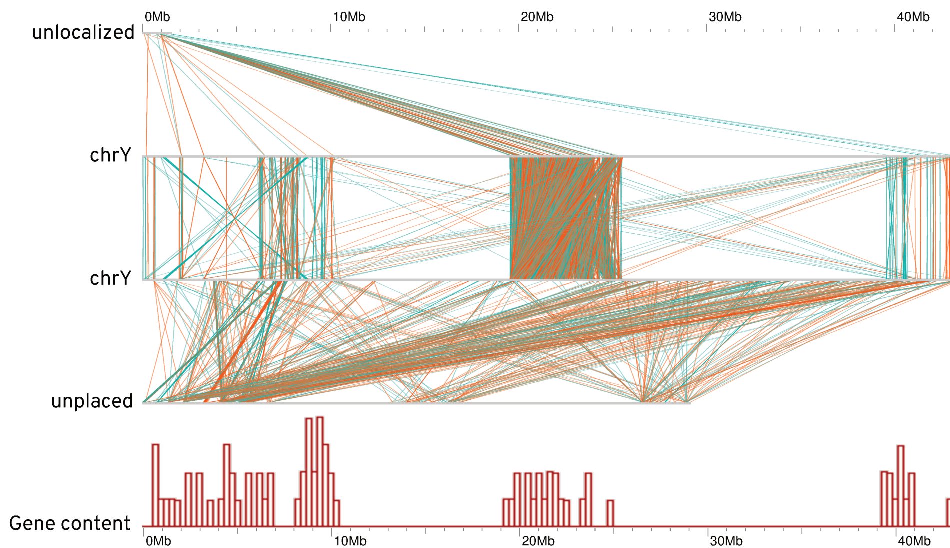


Figure 4.21: Mapping of the similarities (as found by ASGART) between the unplaced sequences, unlocalized sequences, and Y chromosome sequences of *sus scrofa*. Orange duplications are direct, teal are palindromic. Coding gene content profile is extracted from the Ensembl database.

Both from our results and from the previous ones, we can observe for the mammals using a X/Y sex-determination mechanism that for a chromosome, having a high content in duplications is correlated with holding a role in sex determinism. However, some species (*e.g. Gallus gallus*), although possessing a sex-differentiated karyotype (ZZ for males *vs.* ZW for females), have a sex-determining process that is not yet clearly understood. Thus, it may prove interesting to investigate the relationships between the duplicated content of a chromosome and the role it plays in sex-determination. A statistically significant correlation could help understanding the sex-determination mechanisms of species for which it is still unclear, by providing a general indicator of which chromosomes might be involved.

Chapter 5

Perspectives

Although ASGART is now functional, there are still several ways of improvement we would like to pursue, both a from a computer sciences perspective and a biology perspective.

5.1 Computer Sciences

5.1.1 Algorithmic Improvements

Suffix Tree Compression

For now, ASGART is implemented using simple, uncompressed suffix trees. Such structures are ideal as long as enough memory is available, as they allow for a fast access. However, we want to develop ASGART to compare not only two genomes, but many at once, the memory use of suffix arrays will become a consideration.

To alleviate this issue, we plan to implement the FM-index[39] structure following existing work[79] and make this feature available through a command-line switch. Thus, users will be able to make their own choice between the better performances and cache-friendliness of a flat suffix array, *versus* the lower memory use of an FM-index. Moreover, work has already been done on porting string matching using FM-index on FPGA[38]. In our global perspective of trying to improve ASGART performances through hardware acceleration, be it with GPGPU or FPGA, this work would make a good inspiration source.

Better Extension Priming

It has been established[52] that divergences between the arms of a palindromic SD tend to be laid out following an increasing gradient starting from the spacer. By extrapolating this observation to SDs in general, we can expect their duplicons to exhibit a very high identity rate in their middle part, and mostly differ at their extremities.

For now, ASGART is seeding its search with exactly matching k-mers. To improve ASGART precision in adequation with this biological observation, we would like to implement

partial matching in the *Sweep* state with *e.g.* a Levenshtein automaton[112] before going back (for performance concerns) to exactly matching k-mers once the duplicons family detection has been primed. This compromise should allow ASGART to exhibit a better precision on duplicons extremities.

5.1.2 GPGPU

ASGART is currently only using CPUs as a computing unit to do its work. However, GPUs offer great amount of computing power distributed among thousands of cores for relatively low prices when compared to CPU. GPGPU is now a well established domain of applications of these chips and is being largely relied on by, for instance, deep learning, physical simulation or image processing programs. Unfortunately, very few bioinformatics tools make use of this kind of hardware. Therefore, we would like to make part of ASGART algorithm, namely the finite state automaton, run on GPGPU.

This would offer a strong improvement to ASGART for the following motives. First, the exploration of the DNA by the finite state automaton is an intrinsically massively parallel task, as was shown by the nearly trivial implementation of the parallelization on CPU. Moreover, our parallel implementation does not need a lot of synchronization work between the execution threads, and, it does not require any complex operations, as it is basically made of comparison and simple arithmetic operations. Therefore, it should not prove too hard to be ported to GPU.

Secondly, ASGART is practically only CPU-bound during most of its execution, therefore the high parallelization opportunity offered by a GPU should ideally reflect in the wall-clock time; even if each core of a GPU is dramatically less powerful compared to a CPU one, the globally higher flops of a GPGPU should be practically fully exploited thanks to ASGART being massively parallel.

Thirdly, GPGPU clusters are typically cheaper than similar CPU computing clusters with identical computing powers.

As a pure-CPU version remains mandatory, a drawback of a GPU implementation would be the need to maintain two concurrent codebases. Indeed, *kernels* (pieces of code running on the GPU) are either developed using a C-like language, (typically when using OpenCL [132]) or intertwined with standard C or C++ code (although with some restrictions) when using CUDA [25].

5.1.3 UX, UI & Post-Processing

There are several directions in which we could improve the user experience of ASGART, that we detail below.

Graphical Interface

CLI application are only available to people already familiar with the use of a virtual terminal. For the other users, the development of a simple GUI application front-end for ASGART should help. Such an application should naturally be cross-platform, as ASGART itself is multiplatform. However, the performance imperatives should not be as acute as for ASGART itself, as the front-end basically just has to help the user providing the correct set of options and input data to ASGART, launch it, and finally present the progress and result to the user. Therefore, we believe that the use of a language running on the JVM and leveraging the graphical libraries offered by this platform, or using the web platform, should prove to be a good equilibrium between robustness, ease of distribution, and ease of development.

Post Processing

For now, the post-processing stage of ASGART is relatively poor. It only offers two main possibilities; first, the user can fetch ASGART result in one of the supported output formats, namely JSON, GFF2, and GFF3, and use them as they see fit. Second, these results may be fed to the companion tool shipped with ASGART, that can export the duplications families satisfying some filtering criteria and export this set, along with some complementary information, be it as a CSV file, or plotted in one of the available type of graphs. Even though this is a first step, we feel that a far more comprehensive tool suite should allow for smoother interactions between the user, as well as for a more pleasant exploratory work, ideally with interactive capacities.

It could prove useful to extend the not-yet published, prototype-grade application we already have developped (Figure 5.1) into a fully-fledged workshop dedicated to duplications families exploratory research. Desirable functions would be potential storage in a remote relational database to share data with other users; multiple formats of import and export to allow for a smooth interoperation with already existing tools and support of custom tracks visualization to let users integrate *e.g.* gene tracks, masking tracks, transcription tracks, etc.

We identified two main obstacles to the visualization of large datasets. The first lays in the scale of the data. The mere process of rendering so many elements in itself might prove very resource-intensive, and the limited resolution of display devices inevitably leads either to an erasure or overload of details in high scale overviews. The second resides in the visualization system: it is hard to provide a clear overview of huge datasets spanning a large space, such as the billions of basepairs of a genome.

Currently available tools [134, 130, 128, 2, 137] typically tackle these difficulties by offering a relatively tiny small-scale window on the dataset, typically in the order of the few thousands of bases to the few hundreds of thousands of base pairs. It is sometimes combined to a larger-scale view with elided details for coarse browsing. Unfortunately, such small windows are inappropriate for an expressive visualization of datasets such as a duplications families, that tend to spread across a whole chromosome, a whole genome, or even several

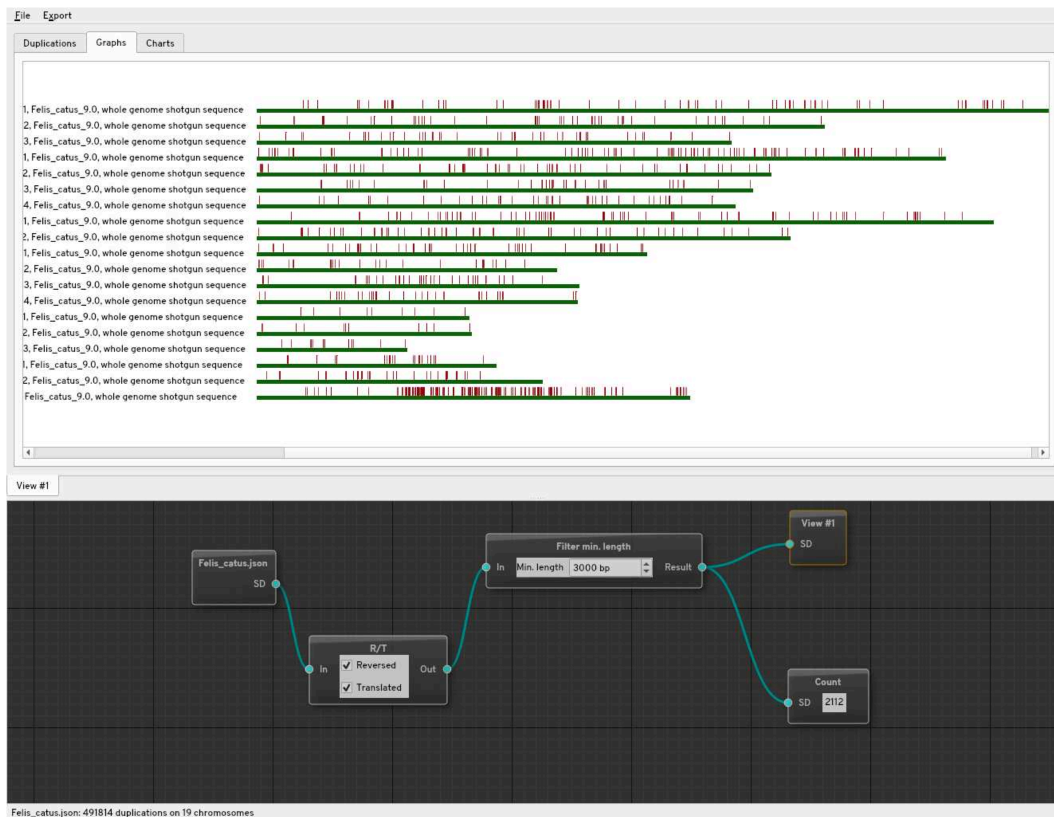


Figure 5.1: A screenshot of the viewer currently being developed along ASGART. At the top, a view of the selected duplicons. At the bottom, a node-based interface[131] allow for the user to easily apply and combine multiple filters on a dataset.

genomes when it comes to multi-species studies. Moreover, in their current implementation, they typically offer a lagging user experience, with high refresh times on every position change.

To improve the current situation, a solution to consider could be the use of a tiling system, akin to map visualization systems such as *e.g.* OpenStreetMaps, combined to a system of progressive level of detail (as used in computer games for a long time). Potentially implemented on GPU, these methods could allow for a smooth, progressive visualization system able to go smoothly and seamlessly from the genome to the base pair scale.

Web Interface

Users and Quotas For now, all jobs are run anonymously. If extremely convenient for the end user, it is a crucial lack of flexibility for the system administrator. There is no way to increase or decrease the data size quota for every user, or to orchestrate the job queue according to a priority system, for instance. There are also practically no available administration or monitoring tool, be it *e.g.* to get an overview of the current working queue, keep the resource usage in check, or manage users. Improving these aspects is a *sine qua none* condition to transform the crude interface of today to a production-grade product ready to be deployed in labs.

Testing Web development is well known to be a brittle landscape. Browsers may tend to interpret creatively the standards of the W3C, the quantity of different browsers and their versions roaming the web is tremendous, and there are always so subtle discrepancies there and there, that can lead to a web application running perfectly on some browser, and failing miserably on some other one. For instance, as we learned during the development of the current result visualization interface, the performances of the SVG rendering engines between Gecko and Blink are uncomfortably divergent, and Safari simply ignores some constraints on the `<input>` HTML element from the HTML5 norm.

Thus, we consider that the current organically developed solution is not (yet) a reliable foundation: it should be corseted by tests; a first test suite to palliate the intrinsic flaws of JavaScript when it comes down to its generous dynamic typing and lack of compile-time errors and warnings; and a second one to ensure a correct rendering on – at least – the most common web rendering engines.

Post-Processing For now, ASGART features two data processing solutions. On the one hand, a post-processing and static visualization tool is distributed with ASGART itself. On the other hand, another visualization application is embedded in the webapp of ASGART. Therefore, there are two incompatible visualization codebases, one in Rust and the second in JavaScript; the first one targeting static SVG files, the second a dynamic, but still SVG-based, web page. The obvious course of action here is to merge together these two suites,

and potentially the hypothetical workshop aforementioned, in a single codebase available to all users and providing all the necessary functions.

5.2 Biology

The current trend in the development of new sequencing technologies is geared toward the production of long, linked reads. By drastically simplifying the assembly of whole genomes compared to the current methods, they should soon make readily available high-quality sequencing of repeated parts of genomes, especially conerning their duplicated parts. Moreover, the process should be both much easier to go through and so for a fraction of the cost of the current alternatives, sharply decreasing the entry cost of duplication studies and furthering its translation from *in-vivo* to *in-silico* methods.

The unlocking of such new datasets should allow for a better understanding of duplicated areas in general, and of SDs in particular; both among and across species. However, the most used today SDs detection and mapping tools are unable to scale to the genome size and over. We developed ASGART to address this issue and now propose a program able to map SDs in assembled sequences up to the multi-genomic scale. In addition to the already previously mentioned uses of SDs, we see several immediate applications for better genomic SDs maps.

5.2.1 Hot Zones Demarcation

In many contexts (*e.g.* population genetics, evolution rate computations, genetic drift estimation or gene selection studies), abnormally active zones of a genome are typically red herrings, skewing the final result while not necessarily adding much information. Therefore, these zones are generally preemptively masked in the concerned DNA fragments before being processed, in order to avoid these issues. For the most well-known genomes, masking annotation tracks are freely available to discriminate these hot zones from their counterpart. For *de novo* masking however, the main solutions are to mask already known sequences (thanks *e.g.* to RepeatMasker and its database) and to remove satellite-like zones, that are known for their dynamism (*e.g.* with TRF).

But these methods do not cover the whole spectrum of hot points: as previously detailed, SDs are known, at least in the mammals, for being strongly correlated with areas of high activity in genomes. And, especially in the primates, they are highly active areas comparatively to single-copy parts of the genome. But due to scale issues, they cannot efficiently be masked neither with existing tools, nor by the use of databases – as SDs sequences vary wildly from species to species. ASGART can be used to detect SDs *de novo* in DNA fragment to mask them, allowing users – when combined with the previously mentioned methods – to establish a more complete map of the stable, single-copy parts of a genome.

This approach has been used, for instance, in a recently published study[107] exploring the discrepancies observed between human mutation rates compared to other primates. AS-

GART is also used towards the same goal in the *Hearvolution* project[1], an ongoing pluridisciplinary study gathering geneticists, anthropologists, mathematicians and computer scientists to establish the main drive of hearing evolution within the primates species.

5.2.2 SDs Dynamics

SDs are also a rich field of studies by themselves, and we envision two main direction of studies we would like to pursue; the first one being an exploration of the enrichment of sex chromosomes in SDs, and the second one concerning the role that SDs may play in understanding gene conversion better.

Haploid Chromosomes & SDs

From a genetic perspective, it has been determined for the human that large palindromic SDs blocks in the Y chromosome act in two ways to palliate the absence of recombination (for the lack of a partner). By actively maintaining a very high identity rate between arms, gene conversion will homogenize duplicated genes located on the arms of a palindrome, and thus create diversity by potentially spreading mutations arising on only one of the arm to the second one. By doing so, gene conversion ensures a parallel evolution of the gene families located on palindromes. In the case of a deleterious mutation, it would be counter-selectioned, and so never transmitted to the second arm. In the end, gene conversion acts as a mean of preserving the integrity of these crucial genes, involved in fertility and cerebral development, and ensure their parallel evolution. Therefore, the question arises to find out whether this mechanism is human-specific, or if other species use a similar method to compensate the lack of recombination on their haploid sex chromosomes.

We would like to determine first if the dynamic observed in the human Y chromosome is particular to the human, or if it can also be found in other mammals sharing the same sex-determination system. In addition to the XY sex-determination systems used by the mammals, other sex-determination systems resulting in a haploid sex chromosome in the karyotypes of the individuals of one of the sex arose in a convergent evolution phenomenon. These other XYs (homogametic females, heterogametic males) and ZWs (heterogametic females, homogametic males) sex-determination systems should expose their individuals to the same problems than the humans. Determining whether these species are relying on the same dynamic to protect the genes their haploid sex chromosome carries or if they created another one would be the next step.

Gene Conversion Characterization

In addition to *e.g.* homologous recombination, gene conversion is a major mechanism to take into account when considering evolutionary mechanisms. Not only does gene conversion

[1]This project is hosted within the UPS/CNRS UMR5288 unit.

differs in result from homologous recombination by spawning asymmetrical genetic material exchanges, it may also takes place in areas shut to other recombination methods, such as *e.g.* the telomeric areas[123]. These two main differences clearly put gene conversion dynamic apart from [non-]homologous recombination one.

However, the erasure of history inherent to gene conversion asymmetrical consequences makes it an elusive phenomenon to study. From the current understanding, it appears that gene conversion takes different characteristics depending on the species concerned, and on the event location within this species genome[52, 83, 42] – be it *e.g.* in its frequency, its typical tract length, or its GC-bias.

As previously detailed, primates SDs are an excellent substrate for gene conversion due to the high homology rate between their duplicons, their content, and their widespread repartition within a genome. As a consequent number of SDs families dating back to the SDs explosion after the split between New-World and Old-World monkeys are shared among primates genomes, building a phylogenetic tree of these SDs families should be envisionable. This tree might prove to be an efficient tool to try reconstruct the history of these sequences, and, implicitly, to establish the profile of gene conversion events that happened between these sequences.

Eventually, such a study would, combined to the existing ones, lay the foundations for a better characterization of gene conversion characteristics within the family of the primates, marking a first milestone before extending the question towards more species.

Что бог, мол, с ними, с генами, бог с ними, хромосомами,
Мы славно поработали и славно отдохнём!

To Hell with these molecules, and genes and chromosomes with them,
We worked well and it is time we take a good rest!

В. Высоцкий, Товарищи учёные (*Comrades Scientists*)

Glossary

***de novo* sequencing**

De novo sequencing refers to the sequencing of a genome *ex nihilo*, without the help of a reference assembly. 26

ABI

Application Binary Interfaces define how binary objects should interact. They are typically needed when objects originally wrote in different languages should communicate. 63

allele

An allele is one of the observed variants of a gene in species. 21

autosome

Autosomes are chromosomes found in pair of similar members. In the cases found in this document, they are synonym with non-sex chromosomes. 92, 93

Bernoulli distribution

A BERNOULLI distribution models a random variable taking either value 1 with a probability p or 0 with probability $1 - p$. 42

bp

bp, for base pairs, is the standard unit to measure the length of DNA sequences. One bp correspond to a nucleotide on each strand of a DNA molecule, or one nucleotide on a denatured DNA strand. 18, 19, 21–23, 28, 29, 32, 34, 38, 39, 41, 46, 47, 52, 53, 56, 59, 64, 69, 73–75, 81, 87–90, 94, 107–109

cache line

Cache lines, or cache blocks, are fixed size blocks that are, for all practical purposes, the smallest addressable unit of a CPU cache. 38

chromatide

In eukaryote genomes, a chromosome is made of two sister chromatides, one from each parent. 23

clade

A clade groups, from a phylogenetic perspective, all species sharing a common ancestor. 16, 92

CLI

Command-Line Interface is a mode of user/program interaction based around the use of a command-based interface inside a standard shell, by opposition to *e.g.* graphical application. 66

CNV

Copy Number Variation are sequences that are found in a different number of copies among the individual of a species. 18

coding sequence

A coding sequence of a genome encode part of a synthesizable. 15, 20

complemented

A complemented sequence has seen all of its nucleotides replaced with their complementary ones, following the A/T and G/C pairs. 20, 30, 53, 65, 69

CUDA

CUDA is a closed, proprietary GPGPU API developed by Nvidia and thus restricted to this brand GPUs; despite these drawbacks, if offers a more comfortable use thanks to the possibility to interleave CPU and GPU code in C++. 98

denatured

A denatured DNA molecule has only one strand left instead of two. 26, 29

duplicon

Duplicon is a name given to the members of a same duplication family. 11, 12, 19, 21, 22, 28, 30–33, 35, 42, 45–47, 52, 53, 59, 64–66, 69, 73, 74, 80, 91–94, 97, 98, 100

E-value

The E-value, or *expected value*, of an alignment is the probability for it to happen by chance between unrelated sequences. 46

exon

Exons are parts of genes that directly code for a part of the protein the gene they belong to encode. 20, 61

FFI

A Foreign Function Interface is a facility of a programming language by which it might call on routines or functions written in another one. 63

format

A format describe how to store a certain kind of information in a digital way. 29, 51–53, 60–62, 65, 66, 68, 69

GC

A Garbage Collector is a runtime component of a programming language automatically handling memory management tasks. 64

GPGPU

General Purpose GPU refers to using a GPU not for graphical purposes, but for more general computational purposes. 98

great ape

See hominoid. 19

HMM

A Hidden Markov Model attempts to statistically modelize a system by assuming it might be represented by a numbers of states linked by transitions. 43

hominoid

Hominoids, or humanoids, or great apes, or hominids, or hominidae, are a family grouping the species of gorillas, chimpanzees, humans and oragutans. 15

homologous

Homologous refer to the same locus, but on the sister chromatide. 16, 22, 23, 48, 88

housekeeping gene

Housekeeping genes are a set of genes required for basic cell machinery and practically ubiquitously expressed. 21

identity rate

The identity rate is a measure, in percent, of how similar two sequences are. 23, 30–33, 40, 45, 46, 52, 53, 73

indel

An indel, short for insertion/deletion, is the addition or the removal of a substring from a string. 33–35, 45, 52, 97

intron

Introns are parts of genes that are not part of the final protein. They typically serve for regulation purposes. 20

LINE

LINEs, for Long INterspersed Elements, are a type of retrotransposons making up to 20% of the human genome. They are typically around 7kbp long. 18, 32

locus

A locus is a precise position within a genome. 18, 23, 53, 109

masked sequence

Masked sequences are typically represented in FASTA file by writing them in lower-case letters. 65

meiose

Meiose is a type of cell-division from which four haploid cells are created from a parent, diploid cell; it is typically used for the creation of gametes. 16

microsatellite

Microsatellites are made of numerous repetitions of a single pattern, ranging from 1 to 5 bp, each with a very high homology rate with the others. 30–33

minisatellite

Minisatellites are made of numerous repetitions of a single pattern, ranging from 5 to 50 bp, each with a very high homology rate with the others. 31

multiFASTA

MultiFASTA is an extension of the FASTA format allowing to store several sequences in a single file. 64

non-allelic

Non-allelic events take place between similar sequences that are not alleles, /i.e./ do not have the same locus. 16, 22, 88

OpenCL

OpenCL is the only open standard for a GPGPU API, designed by the Kronos groups (including but not restricted to Apple, Intel, AMD, Nvidia and Google). 98

orthologous

Orthologous sequences are different sequences, but descending from a common ancestor sequence. 44, 45

OTP

The Open Telecom Platform was the base of the Ericsson telephony offer, and so prioritized stability, high-availability, massive concurrency, code hot-reload and failure tolerance. 67

PAR

The PseudoAutosomal Regions are homologous areas of the X and Y chromosomes where recombination akin to the one happening on autosome pairs still takes place. 21, 92

paralogous

Paralogous sequences are sequences within a species genome that share a common ancestor and diverged from each other. 23

pericentromeric

Pericentromeric sequences are flanking the centromere of a chromosome. 19, 20, 86

poly-A tail

poly-A tails are long sequences of adenine nucleobases sometimes found at the end of coding sequences. 31, 64

polymerization

In this context, polymerization is the reaction during which a DNA strand is built out of free-standing nucleotides. 26, 29

primate

The order of the primates group mostly monkeys and lemuriformes. 19–22, 42, 43

pseudogene

A pseudogene is a degenerated gene that is not expressed anymore, typically due to mutations making it unfunctional. 20, 31, 40

recombination

Recombination is the process by which genetic material is exchanged within the genome. It is the main drive of genetic variation creation. 15, 16

reference genome

A reference genome is well-known genome that is especially studied for various reasons, ranging from historical ones to peculiar characteristics of the species it comes from through being often used as experimentation species. 79

retrotransposon

See transposon. 18, 20

satellite

Satellites are made of successive repetitions of a single pattern, each with a very high homology rate with the others. 31, 61

SIMD

Single-Instruction, Multiple-Data instruction sets expose hardware facilities to apply an operation to several elements at once in a parallel fashion. 38

SINE

SINEs, for Short Interspersed Nuclear Elements, are a type of transposons making up /ca./ 10% of the human genome. They are typically a few hundred bp long. 18, 32

SNP

Single-Nucleotide Polymorphisms are structural variations occuring at a single, known locus. 52

speciation

Speciation is the process giving birth to several species (typically two) from the splitting of a single one. 15

STR

Short Tandem Repeats are DNA sequences made of numerous repeats of a simple pattern typically a few bp long. 31, 33

structural variation

Structural variations are differences in genome that are observed between individuals of the same species. 18, 19, 28, 109

subtelomeric

Subtelomeric sequences are flanking the telomeres of a chromosome. 19, 20

tandem array

A tandem array is a linear repetition of multiple duplicons sharing a very high identity rate. 19, 42

thread

A thread, or execution unit, is a concurrently executed unit of code. On multicore CPUs, the execution is also typically parallel. 47, 62, 65, 86, 98

transposon

Transposons (or /transposable elements/) are sequences of a genome able to change their position in the genome using the cell machinery; whether they do with or without going through a RNA transcription stage splits the transposons from the retro-transposons. 19

unlocalized sequence

Unlocalized sequences are scaffolds (intermediary sequencing results, standing between the read and the assembled sequence) for which the host chromosome is known, but that could not be placed on it. 94, 95

unplaced sequence

Unplaced sequences are scaffolds (intermediary sequencing results, standing between the read and the assembled sequence) for which neither their host chromosome, nor their position in the genome, could be determined. 94, 95

user time

User time is the cumulated amount of time that the CPU(s) spent on a program from its start to its stop. 80, 81, 86

wall-clock time

The wall-clock time is the actual time that a program takes to run from start to stop. 75, 80, 81, 86, 88, 98

ZMW well

Zero-Mode Wavelength wells have a microscopic width and are smaller than visible light wavelength, preventing it to enter the well as it can not finds an electromagnetic mode inside it. 29

Index

ABI, 63
allele, 21
arabidopsis, 80, 88, 91
assembling, 26–28, 35, 41, 44, 60, 61, 94
autosome, 21, 80, 92, 93

cache, 37, 38, 54, 97
chimpanzee, 15, 20, 22, 91
chromatide, 23
clade, 16, 92
CLI, 66
CNV, 18
coding sequence, 15, 20
complemented, 20, 30, 53, 65, 69
complexity, 12, 31, 34–42, 44–46, 58, 59, 98
Cuda, 98

denatured, 26, 29
distance, 30, 32–36, 42, 43, 45, 53, 57, 98
drosophila, 80, 88, 90, 92, 93
duplications, 5, 7, 9, 11, 12, 19, 20, 23, 26, 30–32, 34–36, 39–48, 51–53, 55, 57, 59, 61, 62, 64–66, 68, 69, 71–75, 79, 80, 86–88, 91–95, 97, 99, 101
duplicon, 11, 12, 19, 21, 22, 28, 30–33, 35, 42, 45–47, 52, 53, 59, 64–66, 69, 73, 74, 80, 91–94, 97, 98, 100

evalue, 46
evolution, 9, 11–13, 16, 17, 19, 20, 25, 30, 48, 51, 75, 91, 94
exon, 20, 61

FFI, 63
format, 29, 51–53, 60–62, 65, 66, 68, 69

GC, 64
gene conversion, 22, 23, 48, 91, 94
GFF, 61, 66, 68, 99
gorilla, 15
GPGPU, 98
great ape, 19

haploid, 21, 48, 80
hominoids, 15, 16, 19, 20
homologous, 16, 22, 23, 48, 88
housekeeping gene, 21
human, 11, 12, 15–23, 26, 39, 48, 52, 59, 66, 69, 71, 75, 80, 86, 88, 89, 91–94

identity rate, 23, 30–33, 40, 45, 46, 52, 53, 73, 77
indel, 34, 35, 45, 52, 97
intron, 20

JSON, 61, 66, 68, 99

LINE, 18, 32
locus, 18, 23, 53

mammals, 19–21, 31, 45, 92, 95
masked, 65
meiose, 16
microsatellite, 30–33
minisatellite, 31
mouse, 19, 80, 81, 87–89, 91, 94
multiFASTA, 64

non-allelic, 16, 22, 88

OpenCL, 98
orangutan, 15
orthologous, 44, 45
OTP, 67

palindrome, 21–23, 30, 69, 75, 89–95
paralogous, 23
pericentromeric, 19, 20, 86
phylogeny, 22, 25, 59
poly-A tail, 31, 64
polymerization, 26, 29
primate, 20–22, 42, 43
pseudogene, 20, 40

recombination, 15, 16, 21–23, 25, 48, 88
retrotransposon, 18, 20

satellite, 31, 61
SD, 9, 11–13, 18–22, 25, 30, 32, 33, 40–42, 45, 46, 48, 49, 61, 62, 64, 65, 68, 70, 88–91
sequencing, 11, 12, 15, 23, 25, 26, 28, 41, 48, 60, 62, 79, 80, 91, 93, 94
sex determination, 7, 9, 19–21, 48, 49, 80, 95
SIMD, 38
SINE, 18, 32
SNP, 52
speciation, 15
STR, 31, 33
subtelomeric, 19, 20
suffix array, 37–39, 53, 54, 58, 65, 88, 97
suffix tree, 36–39, 45, 97
sus scrofa, 92, 94, 95

tandem array, 19, 42
thread, 47, 62, 65, 86
transposon, 19

unlocalized, 94, 95
unplaced, 94, 95
user time, 80, 81, 86

variation, 7, 9, 11, 15–20, 23, 28, 29, 41, 48, 80, 92

wall-clock time, 75, 80, 81, 86, 88, 98

X chromosome, 21, 22, 92, 93

Y chromosome, 21, 22, 48, 75, 92–95

zebrafish, 59, 80, 81, 86–88, 90
ZMW, 29
ZW chromosomes, 48, 92, 95

Bibliography

[1] B. Alberts et al. *Molecular Biology of the Cell*. 4th. Garland, 2002.

[2] *AliView*. `https://ormbunkar.se/aliview/`.

[3] Stephen F Altschul et al. "Basic local alignment search tool". In: *Journal of molecular biology* 215.3 (1990), pp. 403–410.

[4] Stephen F Altschul et al. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs". In: *Nucleic acids research* 25.17 (1997), pp. 3389–3402.

[5] Alberto Apostolico et al. "Parallel construction of a suffix tree with applications". In: *Algorithmica* 3.1-4 (1988), pp. 347–365.

[6] Eric Audemard, Thomas Schiex, and Thomas Faraut. "Detecting long tandem duplications in genomic sequences". In: *BMC bioinformatics* 13.1 (2012), p. 83.

[7] Doris Bachtrog. "Y-chromosome evolution: emerging insights into processes of Y-chromosome degeneration". In: *Nature Reviews Genetics* 14.2 (2013), p. 113.

[8] Arturs Backurs and Piotr Indyk. "Edit distance cannot be computed in strongly subquadratic time (unless SETH is false)". In: *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. ACM. 2015, pp. 51–58.

[9] Melanie Bahlo et al. "Recent advances in the detection of repeat expansions with short-read next-generation sequencing". In: *F1000Research* 7 (2018).

[10] Jeffrey A Bailey and Evan E Eichler. "Primate segmental duplications: crucibles of evolution, diversity and disease". In: *Nature Reviews Genetics* 7.7 (2006), p. 552.

[11] Jeffrey A Bailey, Ge Liu, and Evan E Eichler. "An Alu transposition model for the origin and expansion of human segmental duplications". In: *The American Journal of Human Genetics* 73.4 (2003), pp. 823–834.

[12] Jeffrey A Bailey et al. "Recent segmental duplications in the human genome". In: *Science* 297.5583 (2002), pp. 1003–1007.

[13] Jeffrey A Bailey et al. "Segmental duplications: organization and impact within the current human genome project assembly". In: *Genome research* 11.6 (2001), pp. 1005–1017.

[14] Patricia Balaresque et al. “Dynamic nature of the proximal AZFc region of the human Y chromosome: multiple independent deletion and duplication events revealed by microsatellite analysis”. In: *Human mutation* 29.10 (2008), pp. 1171–1180.

[15] Zhirong Bao and Sean R Eddy. “Automated de novo identification of repeat sequence families in sequenced genomes”. In: *Genome research* 12.8 (2002), pp. 1269–1276.

[16] Joseph A Bedell, Ian Korf, and Warren Gish. “MaskerAid: a performance enhancement to RepeatMasker”. In: *Bioinformatics* 16.11 (2000), pp. 1040–1041.

[17] Gary Benson et al. “Tandem repeats finder: a program to analyze DNA sequences”. In: *Nucleic acids research* 27.2 (1999), pp. 573–580.

[18] Guillaume Blanc and Kenneth H Wolfe. “Widespread paleopolyploidy in model plant species inferred from age distributions of duplicate genes”. In: *The plant cell* 16.7 (2004), pp. 1667–1678.

[19] Guillaume Blanc et al. “Extensive duplication and reshuffling in the Arabidopsis genome”. In: *The Plant Cell* 12.7 (2000), pp. 1093–1101.

[20] *CALMIP*. `https://www.calmip.univ-toulouse.fr/`.

[21] Christiam Camacho et al. “BLAST+: architecture and applications”. In: *BMC bioinformatics* 10.1 (2009), p. 421.

[22] Gwo-Liang Chen, Yun-Juan Chang, and Chun-Hway Hsueh. “PRAP: an ab initio software package for automated genome-wide analysis of DNA repeats for prokaryotes”. In: *Bioinformatics* 29.21 (2013), pp. 2683–2689.

[23] Joseph Cheung et al. “Genome-wide detection of segmental duplications and potential assembly errors in the human genome sequence”. In: *Genome biology* 4.4 (2003), R25.

[24] Joseph Cheung et al. “Recent segmental and gene duplications in the mouse genome”. In: *Genome biology* 4.8 (2003), R47.

[25] *CUDA homepage*. `https://developer.nvidia.com/cuda-zone`.

[26] Eva Darai-Ramqvist et al. “Segmental duplications and evolutionary plasticity at tumor chromosome break-prone regions”. In: *Genome research* (2008).

[27] Arthur L Delcher, Steven L Salzberg, and Adam M Phillippy. “Using MUMmer to identify similar regions in large sequence sets”. In: *Current protocols in bioinformatics* 1 (2003), pp. 10–3.

[28] Arthur L Delcher et al. “Alignment of whole genomes”. In: *Nucleic acids research* 27.11 (1999), pp. 2369–2376.

[29] Arthur L Delcher et al. “Fast algorithms for large-scale genome alignment and comparison”. In: *Nucleic acids research* 30.11 (2002), pp. 2478–2483.

[30] Franklin Delehelle et al. "ASGART: fast and parallel genome scale segmental duplications mapping". In: *Bioinformatics* (2018).

[31] Gustave Djedatin et al. "DuplicationDetector, a light weight tool for duplication detection using NGS data". In: *Current Plant Biology* 9 (2017), pp. 23–28.

[32] Kanwaljit S Dulai et al. "The evolution of trichromatic color vision by opsin gene duplication in New World and Old World primates". In: *Genome research* 9.7 (1999), pp. 629–638.

[33] Laurent Duret and Nicolas Galtier. "Biased gene conversion and the evolution of mammalian genomic landscapes". In: *Annual review of genomics and human genetics* 10 (2009), pp. 285–311.

[34] Robert C Edgar and Eugene W Myers. "PILER: identification and classification of genomic repeats". In: *Bioinformatics* 21.suppl_1 (2005), pp. i152–i158.

[35] Evan E Eichler. "Widening the spectrum of human genetic variation". In: *Nature genetics* 38.1 (2006), p. 9.

[36] *EMBOSS Stretcher online manual*. `http://embossgui.sourceforge.net/demo/manual/stretcher.html`.

[37] Martin Farach and S Muthukrishnan. "Optimal logarithmic time randomized suffix tree construction". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 1996, pp. 550–561.

[38] Edward Fernandez, Walid Najjar, and Stefano Lonardi. "String matching in hardware using the FM-Index". In: *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*. IEEE. 2011, pp. 218–225.

[39] Paolo Ferragina and Giovanni Manzini. "Opportunistic data structures with applications". In: *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*. IEEE. 2000, pp. 390–398.

[40] Ivan Flores and George Madpis. "Average binary search length for dense ordered lists". In: *Communications of the ACM* 14.9 (1971), pp. 602–603.

[41] David Fredman et al. "Complex SNP-related sequence variation in segmental genome duplications". In: *Nature genetics* 36.8 (2004), p. 861.

[42] Laura-Jayne Gardiner et al. "Analysis of the recombination landscape of hexaploid bread wheat reveals genes controlling recombination and gene conversion frequency". In: *bioRxiv* (2019), p. 539684.

[43] *GFF2 file format specification*. `http://gmod.org/wiki/GFF2`.

[44] *GFF3 file format specification*. `https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md`.

[45] Giuliana Giannuzzi et al. "Analysis of high-identity segmental duplications in the grapevine genome". In: *BMC genomics* 12.1 (2011), p. 436.

[46] Hani Z Girgis. "Red: an intelligent, rapid, accurate tool for detecting repeats de-novo on the genomic scale". In: *BMC bioinformatics* 16.1 (2015), p. 227.

[47] Stella MK Glasauer and Stephan CF Neuhauss. "Whole-genome duplication in teleost fishes and its evolutionary consequences". In: *Molecular genetics and genomics* 289.6 (2014), pp. 1045–1060.

[48] Sara Goodwin, John D McPherson, and W Richard McCombie. "Coming of age: ten years of next-generation sequencing technologies". In: *Nature Reviews Genetics* 17.6 (2016), p. 333.

[49] Brian J Haas et al. "DAGchainer: a tool for mining segmental genome duplications and synteny". In: *Bioinformatics* 20.18 (2004), pp. 3643–3646.

[50] Tsuyoshi Hachiya et al. "Accurate identification of orthologous segments among multiple genomes". In: *Bioinformatics* 25.7 (2009), pp. 853–860.

[51] Matthew W Hahn, Jeffery P Demuth, and Sang-Gook Han. "Accelerated rate of gene gain and loss in primates". In: *Genetics* (2007).

[52] Pille Hallast et al. "Recombination dynamics of a human Y-chromosomal palindrome: rapid GC-biased gene conversion, multi-kilobase conversion tracts, and rare inversions". In: *PLoS genetics* 9.7 (2013), e1003666.

[53] Richard W Hamming. "Error detecting and error correcting codes". In: *Bell System technical journal* 29.2 (1950), pp. 147–160.

[54] Daniel S. Hirschberg. "A linear space algorithm for computing maximal common subsequences". In: *Communications of the ACM* 18.6 (1975), pp. 341–343.

[55] EB Hook and SG Albright. "Mutation rates for unbalanced Robertsonian translocations associated with Down syndrome. Evidence for a temporal change in New York State live births 1968–1977." In: *American journal of human genetics* 33.3 (1981), p. 443.

[56] Jennifer F Hughes et al. "Chimpanzee and human Y chromosomes are remarkably divergent in structure and gene content". In: *Nature* 463.7280 (2010), p. 536.

[57] Jennifer F Hughes, Helen Skaletsky, and David C Page. "Sequencing of rhesus macaque Y chromosome clarifies origins and evolution of the DAZ (Deleted in AZoospermia) genes". In: *Bioessays* 34.12 (2012), pp. 1035–1044.

[58] Jennifer F Hughes et al. "Sex chromosome-to-autosome transposition events counter Y-chromosome gene loss in mammals". In: *Genome biology* 16.1 (2015), p. 104.

[59] Pablo Ibáñez et al. "α-Synuclein gene rearrangements in dominantly inherited parkinsonism: frequency, phenotype, and mechanisms". In: *Archives of neurology* 66.1 (2009), pp. 102–108.

[60] *Illumina history*. `https://www.illumina.com/science/technology/next-generation-sequencing/illumina-sequencing-history.html`.

[61] *Ion Torrent official page*. `https://www.thermofisher.com/fr/fr/home/life-science/sequencing/next-generation-sequencing/ion-torrent-next-generation-sequencing-technology.html`.

[62] Alec J Jeffreys and Celia A May. "Intense and highly localized gene conversion activity in human meiotic crossover hot spots". In: *Nature genetics* 36.2 (2004), p. 151.

[63] Zhaoshi Jiang et al. "Ancestral reconstruction of segmental duplications reveals punctuated cores of human genome evolution". In: *Nature genetics* 39.11 (2007), p. 1361.

[64] Mark Jobling, Matthew Hurles, and Chris Tyler-Smith. *Human evolutionary genetics: origins, peoples & disease*. Garland Science, 2013.

[65] Matthew E Johnson et al. "Recurrent duplication-driven transposition of DNA during hominoid evolution". In: *Proceedings of the National Academy of Sciences* 103.47 (2006), pp. 17626–17631.

[66] John J Kasianowicz et al. "Characterization of individual polynucleotide molecules using a membrane channel". In: *Proceedings of the National Academy of Sciences* 93.24 (1996), pp. 13770–13773.

[67] Szymon M Kielbasa et al. "Adaptive seeds tame genomic sequence comparison". In: *Genome research* (2011), gr–113985.

[68] Dong Kyue Kim et al. "Linear-time construction of suffix arrays". In: *Annual Symposium on Combinatorial Pattern Matching*. Springer. 2003, pp. 186–199.

[69] Pang Ko and Srinivas Aluru. "Space efficient linear time construction of suffix arrays". In: *Journal of Discrete Algorithms* 3.2-4 (2005), pp. 143–156.

[70] Roman Kolpakov, Ghizlane Bana, and Gregory Kucherov. "mreps: efficient and flexible detection of tandem repeats in DNA". In: *Nucleic acids research* 31.13 (2003), pp. 3672–3678.

[71] Romain Koszul and Gilles Fischer. "A prominent role for segmental duplications in modeling eukaryotic genomes". In: *Comptes Rendus Biologies* 332.2-3 (2009), pp. 254–266.

[72] NV Kovaleva et al. "Results of estimation of mutation rates for translocation trisomy 21". In: *Tsitologiia* 44.11 (2002), pp. 1115–1119.

[73] Ravinesh A Kumar et al. "Recurrent 16p11. 2 microdeletions in autism". In: *Human molecular genetics* 17.4 (2007), pp. 628–638.

[74] Stefan Kurtz and Chris Schleiermacher. “REPuter: fast computation of maximal repeats in complete genomes.” In: *Bioinformatics (Oxford, England)* 15.5 (1999), pp. 426–427.

[75] Stefan Kurtz et al. “Versatile and open software for comparing large genomes”. In: *Genome biology* 5.2 (2004), R12.

[76] Bruce T Lahn and David C Page. “Four evolutionary strata on the human X chromosome”. In: *Science* 286.5441 (1999), pp. 964–967.

[77] Michael J Levene et al. “Zero-mode waveguides for single-molecule analysis at high concentrations”. In: *science* 299.5607 (2003), pp. 682–686.

[78] Victor G Levitsky. “RECON: a program for prediction of nucleosome formation potential”. In: *Nucleic acids research* 32.suppl_2 (2004), W346–W349.

[79] Heng Li. “Fast construction of FM-index for long sequence reads”. In: *Bioinformatics* 30.22 (2014), pp. 3274–3275.

[80] Zhize Li, Jian Li, and Hongwei Huo. “Optimal in-place suffix sorting”. In: *International Symposium on String Processing and Information Retrieval*. Springer. 2018, pp. 268–284.

[81] Cheng Ling and Khaled Benkrid. “Design and implementation of a CUDA-compatible GPU-based core for gapped BLAST algorithm”. In: *Procedia Computer Science* 1.1 (2010), pp. 495–504.

[82] David J Lipman and William R Pearson. “Rapid and sensitive protein similarity searches”. In: *Science* 227.4693 (1985), pp. 1435–1441.

[83] Haoxuan Liu et al. “Tetrad analysis in plants and fungi finds large differences in gene conversion rates but no GC bias”. In: *Nature ecology & evolution* 2.1 (2018), p. 164.

[84] Udi Manber and Gene Myers. “Suffix arrays: a new method for on-line string searches”. In: *siam Journal on Computing* 22.5 (1993), pp. 935–948.

[85] Tomas Marques-Bonet, Santhosh Girirajan, and Evan E Eichler. “The origins and impact of primate segmental duplications”. In: *Trends in Genetics* 25.10 (2009), pp. 443–454.

[86] Edward M McCreight. “A space-economical suffix tree construction algorithm”. In: *Journal of the ACM (JACM)* 23.2 (1976), pp. 262–272.

[87] David T Miller et al. “Microdeletion/duplication at 15q13. 2q13. 3 among individuals with features of autism and other neuropsychiatirc disorders”. In: *Journal of medical genetics* (2008).

[88] Richard C Moore and Michael D Purugganan. “The evolutionary dynamics of plant duplicate genes”. In: *Current opinion in plant biology* 8.2 (2005), pp. 122–128.

[89] Aleksandr Morgulis et al. "A fast and symmetric DUST implementation to mask low-complexity DNA sequences". In: *Journal of Computational Biology* 13.5 (2006), pp. 1028–1040.

[90] Yuta Mori. *The divsufsort library*. `https://github.com/y-256/libdivsufsort`.

[91] *MUMmer online manual*. `http://mummer.sourceforge.net/manual/#identifyingrepeats`.

[92] Claudia Münch et al. "Evolutionary analysis of the highly dynamic CHEK2 duplicon in anthropoids". In: *BMC evolutionary biology* 8.1 (2008), p. 269.

[93] Eugene W Myers and Webb Miller. "Optimal alignments in linear space". In: *Bioinformatics* 4.1 (1988), pp. 11–17.

[94] Saul B Needleman and Christian D Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of molecular biology* 48.3 (1970), pp. 443–453.

[95] Laurent Noé and Gregory Kucherov. "YASS: enhancing the sensitivity of DNA similarity search". In: *Nucleic acids research* 33.suppl_2 (2005), W540–W543.

[96] Pettersson Nyrén, Bertil Pettersson, and Mathias Uhlén. "Solid phase DNA minisequencing by an enzymatic luminometric inorganic pyrophosphate detection assay". In: *Analytical biochemistry* 208.1 (1993), pp. 171–175.

[97] Chris Oehmen and Jarek Nieplocha. "ScalaBLAST: A scalable implementation of BLAST for high-performance data-intensive bioinformatics analysis". In: *IEEE Transactions on Parallel and Distributed Systems* 17.8 (2006), pp. 740–749.

[98] P Raeymaekers et al. "Duplication in chromosome 17p11. 2 in Charcot-Marie-Tooth neuropathy type 1a (CMT 1a)". In: *Neuromuscular disorders* 1.2 (1991), pp. 93–97.

[99] Vinciane Régnier et al. "Emergence and scattering of multiple neurofibromatosis (NF1)-related sequences during hominoid evolution suggest a process of pericentromeric interchromosomal transposition". In: *Human molecular genetics* 6.1 (1997), pp. 9–16.

[100] Sjoerd Repping et al. "Recombination between palindromes P5 and P1 on the human Y chromosome causes massive deletions and spermatogenic failure". In: *The American Journal of Human Genetics* 71.4 (2002), pp. 906–922.

[101] Mark T Ross et al. "The DNA sequence of the human X chromosome". In: *Nature* 434.7031 (2005), p. 325.

[102] Owen A Ross et al. "Genomic investigation of α-synuclein multiplication and parkinsonism". In: *Annals of Neurology: Official Journal of the American Neurological Association and the Child Neurology Society* 63.6 (2008), pp. 743–750.

[103] Anne Rovelet-Lecrux et al. “APP locus duplication causes autosomal dominant early-onset Alzheimer disease with cerebral amyloid angiopathy”. In: *Nature genetics* 38.1 (2006), p. 24.

[104] Steve Rozen et al. “Abundant gene conversion between arms of palindromes in human and ape Y chromosomes”. In: *Nature* 423.6942 (2003), p. 873.

[105] Левенштейн, Владимир Иосифович. “Двоичные коды с исправлением выпадений, вставок и замещений символов”. In: *Доклады Академии наук*. Vol. 163. 4. Российская академия наук. 1965, pp. 845–848.

[106] Kim Rutherford et al. “Artemis: sequence visualization and annotation”. In: *Bioinformatics* 16.10 (2000), pp. 944–945.

[107] Besenbacher S et al. “Direct estimation of mutations in great apes reconciles phylogenetic dating”. In: *Nature Ecology & Evolution* (2019).

[108] Stephan J Sanders et al. “Multiple recurrent de novo CNVs, including duplications of the 7q11. 23 Williams syndrome region, are strongly associated with autism”. In: *Neuron* 70.5 (2011), pp. 863–885.

[109] Frederick Sanger, Steven Nicklen, and Alan R Coulson. “DNA sequencing with chain-terminating inhibitors”. In: *Proceedings of the national academy of sciences* 74.12 (1977), pp. 5463–5467.

[110] Frederick Sanger et al. “Nucleotide sequence of bacteriophage φX174 DNA”. In: *nature* 265.5596 (1977), p. 687.

[111] Aylwyn Scally. “The mutation rate in human evolution and demographic inference”. In: *Current opinion in genetics & development* 41 (2016), pp. 36–43.

[112] Klaus U Schulz and Stoyan Mihov. “Fast string correction with Levenshtein automata”. In: *International Journal on Document Analysis and Recognition* 5.1 (2002), pp. 67–85.

[113] Deepak Sharma et al. “Spectral Repeat Finder (SRF): identification of repetitive sequences using Fourier transformation”. In: *Bioinformatics* 20.9 (2004), pp. 1405–1412.

[114] Andrew J Sharp et al. “Segmental duplications and copy-number variation in the human genome”. In: *The American Journal of Human Genetics* 77.1 (2005), pp. 78–88.

[115] Xinwei She et al. “Mouse segmental duplication and copy number variation”. In: *Nature genetics* 40.7 (2008), p. 909.

[116] Julian Shun and Guy E Blelloch. “A simple parallel cartesian tree algorithm and its application to parallel suffix tree construction”. In: *ACM Transactions on Parallel Computing* 1.1 (2014), p. 8.

[117] Amit U Sinha and Jaroslaw Meller. "Cinteny: flexible analysis and visualization of synteny and genome rearrangements in multiple organisms". In: *BMC bioinformatics* 8.1 (2007), p. 82.

[118] Helen Skaletsky et al. "The male-specific region of the human Y chromosome is a mosaic of discrete sequence classes". In: *Nature* 423.6942 (2003), p. 825.

[119] Arian FA Smit, Robert Hubley, and P Green. *RepeatMasker*. 1996.

[120] Michael Spitzer et al. "VisCoSe: visualization and comparison of consensus sequences". In: *Bioinformatics* 20.3 (2004), pp. 433–435.

[121] Peter H Sudmant et al. "Diversity of human copy number variation and multicopy genes". In: *Science* 330.6004 (2010), pp. 641–646.

[122] Way Sung et al. "Evolution of the insertion-deletion mutation rate across the tree of life". In: *G3: Genes, Genomes, Genetics* (2016), g3–116.

[123] Paul B Talbert and Steven Henikoff. "Centromeres convert but don't cross". In: *PLoS biology* 8.3 (2010), e1000326.

[124] Rick M Tankard et al. "Detecting known repeat expansions with standard protocol next generation sequencing, towards developing a single screening test for neurological repeat expansion disorders". In: *bioRxiv* (2017), p. 157792.

[125] John S Taylor et al. "Comparative genomics provides evidence for an ancient genome duplication event in fish". In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences* 356.1414 (2001), pp. 1661–1679.

[126] John S Taylor et al. "Genome duplication, a trait shared by 22,000 species of ray-finned fish". In: *Genome research* 13.3 (2003), pp. 382–390.

[127] Levi S Teitz et al. "Selection Has Countered High Mutability to Preserve the Ancestral Copy Number of Y Chromosome Amplicons in Diverse Human Lineages". In: *The American Journal of Human Genetics* 103.2 (2018), pp. 261–275.

[128] *The Ensembl Genome Browser*. https://grch37.ensembl.org/Homo_sapiens/Location/Overview?time=1547743185;r=1:1-249250621.

[129] *The jq tool*. https://stedolan.github.io/jq/.

[130] *The NCBI Genome Browser*. https://www.ncbi.nlm.nih.gov/genome/gdv/browser/?context=genome&acc=GCF_000001405.38.

[131] *The nodeeditor library*. https://github.com/paceholder/nodeeditor.

[132] *The OpenCL specification*. https://www.khronos.org/registry/OpenCL/specs/2.2/html/OpenCL_API.html.

[133] *The SQL language specification*. https://www.iso.org/standard/63555.html.

[134] *The UCSC Genome Browser*. http://genome.ucsc.edu/cgi-bin/hgGateway.

[135] *The Vmatch large scale sequence analysis software*. http://vmatch.de/.

[136] Esko Ukkonen. “On-line construction of suffix trees”. In: *Algorithmica* 14.3 (1995), pp. 249–260.

[137] *Unipro UGene Software Suite*. http://ugene.net/.

[138] J Craig Venter et al. “The sequence of the human genome”. In: *science* 291.5507 (2001), pp. 1304–1351.

[139] Natalia Volfovsky, Brian J Haas, and Steven L Salzberg. “A clustering method for repeat analysis in DNA sequences”. In: *Genome Biology* 2.8 (2001), research0027–1.

[140] MS Waterman. “Identification of common molecular subsequence”. In: *Mol. Biol* 147 (1981), pp. 195–197.

[141] Peter Weiner. “Linear pattern matching algorithms”. In: *Switching and Automata Theory, 1973. SWAT’08. IEEE Conference Record of 14th Annual Symposium on*. IEEE. 1973, pp. 1–11.

[142] Maisa Yoshimoto et al. “PTEN genomic deletions that characterize aggressive prostate cancer originate close to segmental duplications”. In: *Genes, Chromosomes and Cancer* 51.2 (2012), pp. 149–160.

[143] Zheng Zhang et al. “A greedy algorithm for aligning DNA sequences”. In: *Journal of Computational biology* 7.1-2 (2000), pp. 203–214.

[144] Qian Zhao et al. “Segmental duplications in the silkworm genome”. In: *BMC genomics* 14.1 (2013), p. 521.