A Pseudo-Boolean programming approach for computing the breakpoint distance between two genomes with duplicate genes

Sébastien Angibaud, Guillaume Fertin, Irena Rusu, Annelyse Thevenin, Stéphane Vialette

A Pseudo-Boolean programming approach for computing the breakpoint distance between two genomes with duplicate genes

Sébastien Angibaud[1], Guillaume Fertin[1], Irena Rusu[1], Annelyse Thévenin[2], and Stéphane Vialette[2]

[1] Laboratoire d'Informatique de Nantes-Atlantique (LINA), FRE CNRS 2729
Université de Nantes, 2 rue de la Houssinière, 44322 Nantes Cedex 3 - France
{angibaud,fertin,rusu}@lina.univ-nantes.fr
[2] Laboratoire de Recherche en Informatique (LRI), UMR CNRS 8623
Faculté des Sciences d'Orsay - Université Paris-Sud, 91405 Orsay - France
{thevenin,vialette}@lri.fr

Abstract. Comparing genomes of different species has become a crucial problem in comparative genomics. Recent research have resulted in different genomic distance definitions: number of breakpoints, number of common intervals, number of conserved intervals, Maximum Adjacency Disruption number (MAD), etc. Classical methods (usually based on permutations of gene order) for computing genomic distances between whole genomes are however seriously compromised for genomes where several copies of the same gene may be scattered across the genome. Most approaches to overcoming this difficulty are based on the *exemplar* method (keep exactly one copy in each genome of each duplicated gene) and the *maximum matching* method (keep as many copies as possible in each genome of each duplicated gene). Unfortunately, it turns out that, in presence of duplications, most problems are **NP**–hard, and hence several heuristics have been recently proposed.
Extending research initiated in [2], we propose in this paper a novel generic pseudo-boolean approach for computing the exact breakpoint distance between two genomes in presence of duplications for both the *exemplar* and *maximum matching* methods. We illustrate the application of this methodology on a well-known public benchmark dataset of γ-Proteobacteria.

Keywords: genome rearrangement, duplication, breakpoint distance, heuristic, pseudo-boolean programming.

1 Introduction

The order of genes in the genomes of species can change during evolution and can provide information about their phylogenetic relationship. Two main approaches are possible. The first one consists in using different types of rearrangement operations and to find possible rearrangement scenarios using these operations (one

of the most common rearrangement operations is reversals, which reverse the order of a subset of neighboring genes) [11]. The second one consists in computing a (dis-)similarity measure based on the gene order and most common rearrangement operations [15,8,4,1]. We focus in this paper on the latter approach.

Several similarity (or dissimilarity) measures between two whole genomes have been recently proposed, such as the number of breakpoints [15,8,4], the number of reversals [8,11], the number of conserved intervals [6], the number of common intervals [7], the Maximum Adjacency Disruption Number (MAD) [16], *etc.* However, in the presence of duplications and for each of the above measures, one has first to disambiguate the data by inferring orthologs, *i.e.*, a non-ambiguous mapping between the genes of the two genomes. Up to now, two extremal approaches have been considered : the *exemplar* model and the *maximum matching* model. In the *exemplar* model [15], for all gene families, all but one occurrence in each genome is deleted. In the *maximum matching* model [4,10], the goal is to map as many genes as possible. These two models can be considered as the extremal cases of the same generic homolog assignment approach.

Unfortunately, it has been shown that, for each of the above mentioned measures, whatever the considered model (*exemplar* or *maximum matching*), the problem becomes **NP**–complete as soon as duplicates are present in genomes [8,4,6,10] ; a few inapproximability results are known for some special cases. Therefore, several heuristic methods have been recently devised to obtain (hopefully) good solutions in a reasonable amount of time [5,7]. However, while it is relatively easy to compare heuristics between them, until now very little is known about the absolute accuracy of these heuristics. Therefore, there is a great need for algorithmic approaches that compute exact solutions for these genomic distances.

Extending research initiated in [2], we propose in this paper a novel generic pseudo-boolean approach for computing the exact breakpoint distance between two genomes in presence of duplications for both the *exemplar* and *maximum matching* methods. Furthermore, we show strong evidence that a fast and simple heuristic based on iteratively finding longest common subsequences provides very good results on our dataset of γ-Proteobacteria.

This paper is organized as follows. In Section 2, we present some preliminaries and definitions. We focus in Section 3 on the problem of finding the minimum number of breakpoints under the two models and we give a pseudo-boolean program together with some reduction rules. Section 4 is devoted to experimental results on a dataset of γ-Proteobacteria.

2 Preliminaries

From an algorithmic perspective, a *unichromosomal genome* is a signed sequence over a finite alphabet, referred hereafter as the alphabet of *gene families*. Each element of the sequence is called a *gene*. DNA has two strands, and genes on a genome have an orientation that reflects the strand of the genes. We represent the order and directions of the genes on each genome as a sequence of signed

elements, *i.e.*, elements with signs "+" and "−". Let G_0 and G_1 be two genomes. For each $x \in \{0, 1\}$, we denote the label at position i in G_x by $G_x[i]$, $1 \leq i \leq n_x$, and we write n_x for the number of genes in genome G_x and $\mathsf{occ}_x(\mathbf{g}, i, j)$ for the number of genes $\mathbf{g}$ (and $-\mathbf{g}$) in G_x between positions i and j, $1 \leq i \leq j \leq n_x$. To simplify notations, we abbreviate $\mathsf{occ}_x(\mathbf{g}, 1, n_x)$ to $\mathsf{occ}_x(\mathbf{g})$.

In order to deal with the inherent ambiguity of duplicated genes, we now precisely define what is a *matching* between two genomes. Roughly speaking, a matching between two genomes can be seen as a way to describe a putative assignment of orthologous pairs of genes between the two genomes (see for example [11]). A matching $\mathcal{M}$ between genomes G_0 and G_1 is a set of pairwise disjoint pairs $(G_0[i], G_1[j])$, where $G_0[i]$ and $G_1[j]$ belong to the same gene family regardless of the sign, *i.e.*, $|G_0[i]| = |G_1[j]|$. Genes of G_0 and G_1 that belong to a pair of the matching $\mathcal{M}$ are said to be *saturated* by $\mathcal{M}$, or $\mathcal{M}$-saturated for short. A matching $\mathcal{M}$ between G_0 and G_1 is said to be *maximum* if for any gene family, there are no two genes of this family that are unmatched for $\mathcal{M}$ and belong to G_0 and G_1, respectively.

The above definition allows us a large degree of freedom in the choice of the matching between two genomes. Two types of matching are usually considered and specify the underlying model to focus on for computing the desired genomic distance. In the *exemplar* model, the matching $\mathcal{M}$ is required to saturate exactly one gene of each gene family, *i.e.*, the size of the matching is the number of gene families. In the *maximum matching* model, the matching $\mathcal{M}$ is required to saturate as many genes of any gene family as possible, *i.e.*, $\mathcal{M}$ is a matching of maximum cardinality. Let $\mathcal{M}$ be any matching between G_0 and G_1 that fulfills the requirements of a given model (*exemplar* or *maximum matching*). By first deleting non-saturated genes and next renaming genes in G_0 and G_1 according to the matching $\mathcal{M}$, we may now assume that both G_0 and G_1 are duplication-free, *i.e.* G_1 is a signed permutation of G_0. We call the resulting genomes $\mathcal{M}$-*pruned*.

Let G_0 and G_1 be two duplication-free genomes of size n. Without loss of generality, we may assume that G_0 is the identity permutation, *i.e.*, $G_0 = 1\ 2\ \ldots\ n$. We say that there is a *breakpoint* after gene $G_0[i]$, $1 \leq i < n$, in G_0 if neither $G_0[i]$ and $G_0[i+1]$ nor $-G_0[i+1]$ and $-G_0[i]$ are consecutive genes in G_1, otherwise we say that there is an *adjacency* after gene $G_0[i]$. For example, if $G_0 = 1\ 2\ 3\ 4\ 5$ and $G_1 = 1\ -3\ -2\ 4\ 5$, then we have a breakpoint in G_0 after genes 1 and 3 (and hence we have an adjacency in G_0 after genes 2 and 4).

Let G_0 and G_1 be two genomes and $\mathcal{M}$ be a matching under any model (*exemplar* or *maximum matching*) between G_0 and G_1. We define $A_{\mathcal{M}}(G_0, G_1)$ and $B_{\mathcal{M}}(G_0, G_1)$ to be the number of adjacencies and the number of breakpoints between the two $\mathcal{M}$-pruned genomes.

We are now in position to formally define the optimization problem we are interested in. Given two genomes G_0 and G_1 and a model (*exemplar* or *maximum matching*), find a matching $\mathcal{M}$ between G_0 and G_1 that fulfills the requirements of the model such that the number of breakpoints between the two $\mathcal{M}$-pruned genomes is as small as possible.

3 An exact algorithm

3.1 Pseudo-boolean problem

Minimizing the number of breakpoints between two genomes with duplications is an **NP**–hard problem under the *exemplar* model even when $\mathtt{occ}_0(\mathbf{g}) = 1$ for all genes $\mathbf{g}$ in G_0 and $\mathtt{occ}_1(\mathbf{g}) \leq 2$ for all genes $\mathbf{g}$ in G_1 [8]. Consequently, the NP-hardness also holds under the *maximum matching* model.

The exact algorithms we define in this section attempt to take advantage of the existing solvers, and more precisely of the linear pseudo-boolean solvers, which are a generalization of the SAT solvers. To this end, we have to express our problem (with its two variants, according to the *exemplar* or *maximum matching* model) as a linear pseudo-boolean problem (or LPB problem), *i.e.* as a linear program [17] whose variables take 0 or 1 values. A number of generalizations of SAT solvers to `LPB` solvers have been proposed (`Pueblo` [18], `Galena` [9], `OPBDP` [3] and more). We decided to use for our tests the `minisat+` LPB solver [12] because of its good results during PB evaluation 2005 (special track of the SAT COMPETITION 2005).

Instead of directly writing a program that minimizes the number of breakpoints, we chose to write the complementary program which consists in maximizing the number of adjacencies between the two given genomes. There are two reasons for this choice. First, the constraints are simpler and less numerous in this latter case ; moreover, experimental tests moreover showed that the running time of our program is noticeably better by focusing on adjacencies. Second, it is easy to notice that minimizing the number of breakpoints and maximizing the number of adjacencies are equivalent problems under both the *exemplar* and *maximum matching* models. Indeed, according to the above notations, given a matching $\mathcal{M}$ between two genomes G_0 and G_1 we have:

$$B_{\mathcal{M}}(G_0, G_1) + A_{\mathcal{M}}(G_0, G_1) = |\mathcal{M}| - 1. \tag{1}$$

For the *exemplar* and *maximum matching* models, all the matchings $\mathcal{M}$ satisfying the model have the same size, and hence $B_{\mathcal{M}}(G_0, G_1) + A_{\mathcal{M}}(G_0, G_1)$ is a constant. Therefore, maximizing $A_{\mathcal{M}}(G_0, G_1)$ is equivalent to minimizing $B_{\mathcal{M}}(G_0, G_1)$.

3.2 Maximizing the number of adjacencies

The `LPB` program we propose considers two genomes with duplications and performs an $\mathcal{M}$-pruning which maximizes the number of adjacencies according to a specified model (*exemplar* or *maximum matching*). As discussed above, the resulting matching also minimizes the number of breakpoints between the two genomes. The `LPB` program, Program `Breakpoint-Maximum-Matching`, for the *maximum matching* model is given in Figure 1. The *exemplar* variant is easily obtained by performing only a few changes that are discussed subsequently.

Program `Breakpoint-Maximum-Matching`

Objective :

$$\texttt{Maximize} \sum_{0 \leq i < n_0} \sum_{i < j \leq n_0} \sum_{0 \leq k < n_1} \sum_{k < \ell \leq n_1} d(i,j,k,\ell)$$

Constraints :

(C.01) $\forall\ 1 \leq i \leq n_0, \sum_{\substack{1 \leq k \leq n_1 \\ |G_0[i]|=|G_1[k]|}} a(i,k) = b_0(i)$

$\forall\ 1 \leq k \leq n_1, \sum_{\substack{1 \leq i \leq n_0 \\ |G_0[i]|=|G_1[k]|}} a(i,k) = b_1(k)$

(C.02) $\forall\ 0 \leq x \leq 1, \forall\ \mathbf{g} \in \mathcal{G}, \sum_{\substack{1 \leq i \leq n_x \\ |G_x[i]|=|\mathbf{g}|}} b_x(i) = \min(\mathsf{occ}_0(\mathbf{g}), \mathsf{occ}_1(\mathbf{g}))$

(C.03) $\forall\ 0 \leq x \leq 1, \forall\ 1 \leq i \leq j-1 < n_x,\ c_x(i,j) + \sum_{i<p<j} b_x(p) \geq 1$

(C.04) $\forall\ 0 \leq x \leq 1, \forall\ 1 \leq i < p < j \leq n_x,\ c_x(i,j) + b_x(p) \leq 1$

(C.05) $\forall\ 1 \leq i < j \leq n_0, \forall\ 1 \leq k < \ell \leq n_1,$
such that $G_0[i] = G_1[k]$ and $G_0[j] = G_1[\ell]$,
$a(i,k) + a(j,\ell) + c_0(i,j) + c_1(k,\ell) - d(i,j,k,\ell) \leq 3$

(C.06) $\forall\ 1 \leq i < j \leq n_0, \forall\ 1 \leq k < \ell \leq n_1,$
such that $G_0[i] = G_1[k]$ and $G_0[j] = G_1[\ell]$,
$a(i,k) - d(i,j,k,\ell) \geq 0$
$a(j,\ell) - d(i,j,k,\ell) \geq 0$
$c_0(i,j) - d(i,j,k,\ell) \geq 0$
$c_1(k,\ell) - d(i,j,k,\ell) \geq 0$

(C.07) $\forall\ 1 \leq i < j \leq n_0, \forall\ 1 \leq k < \ell \leq n_1,$
such that $G_0[i] = -G_1[\ell]$ `and` $G_0[j] = -G_1[k]$,
$a(i,\ell) + a(j,k) + c_0(i,j) + c_1(k,\ell) - d(i,j,k,\ell) \leq 3$

(C.08) $\forall\ 1 \leq i < j \leq n_0,\ \forall\ 1 \leq k < \ell \leq n_1,$
such that $G_0[i] = -G_1[\ell]$ `and` $G_0[j] = -G_1[k]$,
$a(i,\ell) - d(i,j,k,\ell) \geq 0$
$a(j,k) - d(i,j,k,\ell) \geq 0$
$c_0(i,j) - d(i,j,k,\ell) \geq 0$
$c_1(k,\ell) - d(i,j,k,\ell) \geq 0$

(C.09) $\forall\ 1 \leq i < j \leq n_0,\ \forall\ 1 \leq k < \ell \leq n_1,$
such that $\{|G_0[i]|, |G_0[j]|\} \neq \{|G_1[k]|, |G_1[\ell]|\}$ `or` $G_0[i] - G_0[j] \neq G_1[k] - G_1[\ell]$,
$d(i,j,k,\ell) = 0$

(C.10) $\forall\ 1 \leq i < j \leq n_0,$
$\sum_{1 \leq k < n_1} \sum_{k < \ell \leq n_1} d(i,j,k,\ell) \leq 1$

Domains :

$\forall\ x \in \{0,1\}, \forall\ 1 \leq i < j \leq n_0, \forall\ 1 \leq k < \ell \leq n_1,$
$a(i,k),\ b_x(i),\ c_x(i,k),\ d(i,j,k,\ell) \in \{0,1\}$

Fig. 1. Program `Breakpoint-Maximum-Matching` for finding the maximum number of adjacencies between two genomes under the *maximum matching* model.

Program `Breakpoint-Maximum-Matching` considers two genomes G_0 and G_1 of respective lengths n_0 and n_1. The objective function, the variables and the constraints are briefly discussed hereafter.

Variables:

- Variables $a(i,k)$, $1 \leq i \leq n_0$ and $1 \leq k \leq n_1$, define a matching $\mathcal{M}$: $a_{i,k} = 1$ if and only if the gene at position i in G_0 is matched with the gene at position k in G_1 in $\mathcal{M}$.
- Variables $b_x(i)$, $x \in \{0,1\}$ and $1 \leq i \leq n_x$, represent the $\mathcal{M}$-*saturated* genes: $b_x(i) = 1$ if and only if the gene at position i in G_x is saturated by the matching $\mathcal{M}$. Clearly, $\sum_{1 \leq i \leq n_0} b_0(i) = \sum_{1 \leq k \leq n_1} b_1(k)$, and this is precisely the size of the matching $\mathcal{M}$.
- Variables $c_x(i,j)$, $x \in \{0,1\}$ and $1 \leq i < j \leq n_x$, represent *consecutive genes* according to the matching $\mathcal{M}$: $c_x(i,j) = 1$ if and only if the genes at positions i, j in G_x are saturated by $\mathcal{M}$ and no gene at position p, $i < p < j$, is saturated by $\mathcal{M}$.
- Variables $d(i,j,k,\ell)$, $1 \leq i < j \leq n_0$ and $1 \leq k < \ell \leq n_1$, represent *adjacencies* according to the matching $\mathcal{M}$: $d(i,j,k,\ell) = 1$ if and only if (i) either $(G_0[i], G_1[k])$ and $(G_0[j], G_1[\ell])$ are two edges of $\mathcal{M}$, or $(G_0[i], G_1[\ell])$ and $(G_0[j], G_1[k])$ are two edges of $\mathcal{M}$, (ii) $G_0[i]$ and $G_0[j]$ are consecutive in G_0 according to $\mathcal{M}$, (iii) $G_1[k]$ and $G_1[\ell]$ are consecutive in G_1 according to $\mathcal{M}$.

Objective function:

The objective of Program `Breakpoint-Maximum-Matching` is to maximize the number of adjacencies between the two considered genomes. This objective reduces in our model to maximizing the sum of all variables $d(i,j,k,\ell)$.

Constraints:

Assume $x \in \{0,1\}$, $1 \leq i < j \leq n_0$ and $1 \leq k < \ell \leq n_1$.

- Constraint **(C.01)** ensures that each gene of G_0 and of G_1 is matched at most once, *i.e.*, $b_0(i) = 1$ (resp. $b_1(k) = 1$) if an only if gene i (resp. k) is matched in G_0 (resp. G_1) ; see Figure 2 for an illustration. Moreover, the matching is possible only between genes in the same family. It is worth noticing here that we do not specifically ask that $a(i,k) = 0$ when i and k concern genes belonging to different families. This is simply not necessary.
- Constraint **(C.02)** defines the model (*i.e.* the *maximum matching* model, in this case). For each gene family $\mathbf{g}$, one must have a single matched gene for the *exemplar* model and $min(\mathsf{occ}_0(\mathbf{g}), \mathsf{occ}_1(\mathbf{g}))$ matched genes for the *maximum matching* model (see Figure 2).
- Constraints in **(C.03)** and **(C.04)** express the definition of consecutive genes, thus fixing the values of the variables c_x. The variable $c_x(i,j)$ is equal to 1 if and only if there exists no p such that $i < p < j$ and $b_x(p) = 1$. Again, it is worth noticing that the constraints do not force the variables $c_x(i,j)$ to have exactly the values we intuitively wish according to the abovementioned

interpretation. Here, we accept that $c_x(i,j) = 1$ even if the gene at position i or j is *not* matched. However, this will pose no problem in the sequel.

- Constraints in **(C.05)** to **(C.10)** define variables d. In the case where $G_0[i] = G_1[k]$ and $G_0[j] = G_1[\ell]$, constraints **(C.05)** and **(C.06)** ensure that we have $d(i,j,k,\ell) = 1$ if and only if all variables $a(i,k)$, $a(j,\ell)$, $c_0(i,j)$ and $c_1(k,\ell)$ are equal to 1. In the case where $G_0[i] = -G_1[\ell]$ and $G_0[j] = -G_1[k]$, constraints **(C.07)** and **(C.08)** ensure that we have $d(i,j,k,\ell) = 1$ if and only if all variables $a(i,\ell)$, $a(j,k)$, $c_0(i,j)$ and $c_1(k,\ell)$ are equal to 1. Constraint **(C.09)** fixes the variable $d(i,j,k,\ell)$ to 0 if none of the two cases above holds. Constraint **(C.10)** requires to have at most one adjacency for every pair (i,j). See Figure 3 for a simple illustration.

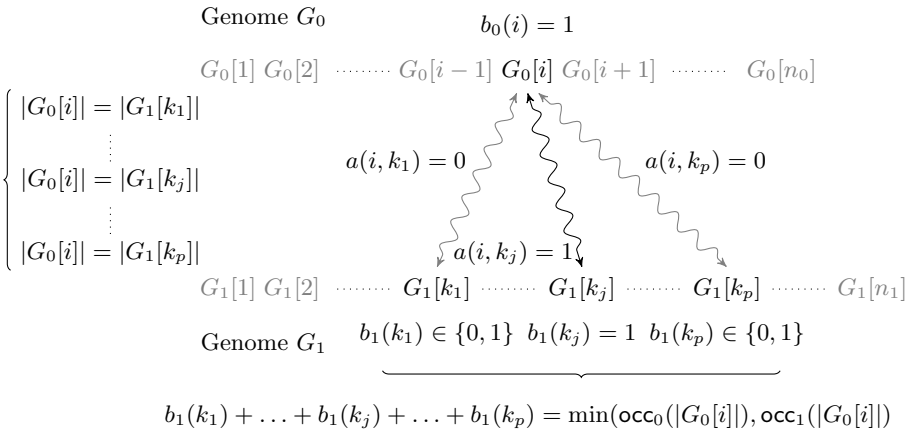

Fig. 2. Illustration of the constraints on variable $b_0(i)$, $1 \leq i \leq n_0$. If gene $G_0[i]$ appears in positions $k_1 < k_2 < \ldots < k_p$ in G_1 and gene $G_0[i]$ is mapped to gene $G_1[k_j]$ in the solution mapping, then (i) $a(i,k_j) = 1$, *i.e.*, gene $G_0[i]$ is mapped to gene $G_1[k_j]$, (ii) $a(i,k_q) = 0$ for $1 \leq q \leq p$ and $q \neq j$, *i.e.*, gene $G_0[i]$ is mapped to only one gene in G_1, (iii) $b_0(i) = 1$, *i.e.*, gene $G_0[i]$ is mapped to a gene of G_1 and (iv) $b_1(k_j) = 1$, *i.e.*, gene $G_1[k_j]$ is mapped to a gene of G_0. Observe that one may have in addition $b_1(k_q) = 1$ for some $1 \leq q \leq p$ and $q \neq j$ if $\min(\mathsf{occ}_0(|G_0[i]|), \mathsf{occ}_1(|G_0[i]|) \geq 1$ (this observation is however no longer valid for the *exemplar* model).

Program `Breakpoint-Maximum-Matching` has $O((n_0 n_1)^2)$ constraints and $O((n_0 n_1)^2)$ variables, which could result in a time-consuming computation. Several simple rules have been used in order to speed-up the execution, some of which help to reduce the number of variables and constraints. They are discussed in the next subsection.

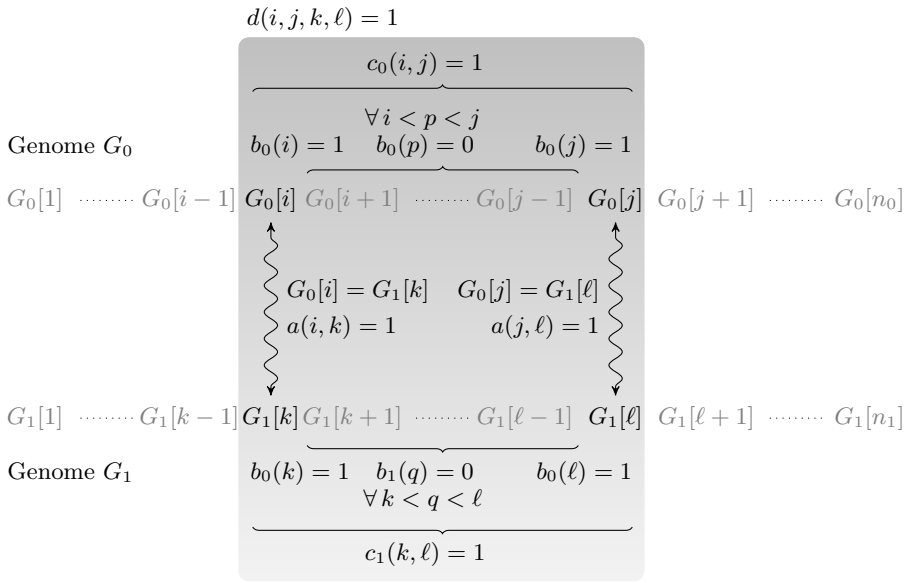

Fig. 3. Illustration of the constraints on variable $d(i,j,k,\ell)$, $1 \leq i < j \leq n_0$ and $1 \leq k < \ell \leq n_1$, for $G_0[i] = G_1[k]$ and $G_0[j] = G_1[\ell]$. The two genes $G_0[i]$ and $G_[j]$ are adjacent according to a solution mapping if there exist two genes $G_1[k]$ and $G_1[\ell]$, $G_0[i] = G_1[k]$ and $G_0[j] = G_1[\ell]$, such that (i) $G_0[i]$ is mapped to $G_1[k]$, *i.e.*, $a(i,k) = 1$, (ii) $G_0[j]$ is mapped to $G_1[\ell]$, *i.e.*, $a(j,\ell) = 1$, (iii) no gene between $G_0[i]$ and $G_0[j]$ is mapped to a gene of G_1, *i.e.*, $c_0(i,j) = 1$ and (iv) no gene between $G_1[k]$ and $G_1[\ell]$ is mapped to a gene of G_1, *i.e.*, $c_1(k,\ell) = 1$. The above situation reduces in our modelization to $d(i,j,k,\ell) = 1$.

3.3 Speeding-up the program

We briefly describe in this section some rules for speeding-up the pseudo-boolean program.

Pre-processing the genomes. The genomes are pairwise pre-processed to delete all genes that do not appear in both genomes. For the *exemplar* model, consecutive occurrences of a gene (with the same sign) are reduced to only one occurrence to this gene. For the γ-proteobacteria benchmark set, the average size of a genome reduces from 3000 to 1300.

Reducing the number of variables and constraints. Due to space constraints we only list few easy reduction rules. For non-duplicated genes, *i.e.*, $\mathsf{occ}_0(g) = \mathsf{occ}_1(g) = 1$, the corresponding variable $a_{i,k}$ is set directly to 1, as well as the two variables $b_0(i)$ and $b_1(k)$. Also, if two non-duplicated genes occur consecutively or in reverse order with opposite signs, the corresponding variable $d()$ is set directly to 1 and the related constraints are discarded. For the *exemplar* model,

we must have exactly one occurrence of each gene in each genome, and hence if the same gene occurs, say in G_0, at positions i and j, then the corresponding variable $d()$ is set directly to 0 and the related constraints are discarded. If for two genes, say occurring at positions i and j in G_0 and k and ℓ in G_1, at least one gene occurring between position i and j in G_0 or k and ℓ in G_1 must be saturated in any matching $\mathcal{M}$, then the corresponding variable $d(i, j, k, \ell)$ is set directly to 0 and the related constraints are discarded (details omitted).

Adding redundancy. While adding redundancy to a pseudo-boolean program is certainly useless from a correctness point of view, it can however have a major impact on the practical performance of the programs. For example, Program `Breakpoint-Maximum-Matching` contains some redundant constraints (**(C.06)**, **(C.08)** and **(C.10)**) that significantly improved the running time of the program.

4 Experimental results

Thanks to the `LPB` program discussed previously, as well as formula (1), we are now able to determine the minimum number of breakpoints between pairs of genomes that contain duplicates. This minimum number of breakpoints will be computed according to the two above mentioned models, i.e. the *exemplar* and *maximum matching* models.

To this end, we used a dataset of γ-proteobacteria genomes, originally studied in [13], and exploited several times since then. This dataset is composed of twelve complete linear genomes of γ-Proteobacteria out of the thirteen originally studied in [13]. Indeed, the thirteenth genome (*V.cholerae*) was not considered, since it is composed of two chromosomes, and hence does not fit in the model we considered here for representing genomes. More precisely, the dataset is composed of the genomes of the following species:

- *Buchnera aphidicola APS* (`Baphi`, Genbank accession number NC_002528),
- *Escherichia coli K12* (`Ecoli`, NC_000913),
- *Haemophilus influenzae Rd* (`Haein`, NC_000907),
- *Pseudomonas aeruginosa PA01* (`Paeru`, NC_002516),
- *Pasteurella multocida Pm70* (`Pmult`, NC_002663),
- *Salmonella typhimurium LT2* (`Salty`, NC_003197),
- *Xanthomonas axonopodis pv. citri 306* (`Xaxon`, NC_003919),
- *Xanthomonas campestris* (`Xcamp`, NC_0 03902),
- *Xylella fastidiosa 9a5c* (`Xfast`, NC_002488),
- *Yersinia pestis CO_92* (`Ypest-CO92`, NC_003143),
- *Yersinia pestis KIM5 P12* (`Ypest-KIM`, NC_004088) and
- *Wigglesworthia glossinidia brevipalpis* (`Wglos`, NC_004344).

The computation of a partition of the complete set of genes into gene families, where each family is supposed to represent a group of homologous genes, is taken from [5] (this partition was actually provided to these authors by Lerat [13]). It

should be noted that in average, 11% of duplicated genes are present in these genomes.

The `LPB` engine is powered by `minisat+` [12]. Computations were carried out on a Quadri Intel(R) Xeon(TM) CPU 3.00 GHz with 16Gb of memory running under Linux. Under the *maximum matching* model, `minisat+` runs our program `Breakpoint-Maximum-Matching` (implemented using the speeding-up rules described in Section 3.3) in less than 10s for 56 out of the 66 possible pairs of genomes, and in several minutes for the remaining 10 pairs. The results are provided in Table 1.

The first conclusion that can be drawn from these results is the following: the pseudo-boolean approach we have considered here is a good approach for computing the minimum number of breakpoints for the *maximum matching* model, since *all* the results have been obtained within a few minutes. However, as already observed in [1] for maximizing the number of *common intervals* between two genomes, we notice that the *exemplar* model is the main bottleneck of our approach. Indeed, for the *exemplar* model, only 49 out of 66 (that is about 74%) results have been obtained within a few minutes (we stopped the computation of the 17 remaining cases after a few days). We still have no formal explanation for this surprising and counter-intuitive fact. The 49 results we have obtained are given in Table 2.

Besides the fact that computing the minimum number of breakpoints under the *maximum matching* model proves to be feasible under our pseudo-boolean approach, we find interesting to note that we have a sufficient number of results in both the *maximum matching* and the *exemplar* models to test the absolute accuracy of possible heuristics for these two problems. Indeed, if one wishes to obtain fast (though not optimal) results by using a given heuristic, it is relevant to know how tight this heuristic is. We carried out this study, focusing on two heuristics (one for the *maximum matching* model, the other for the *exemplar* model), that are both based on iteratively choosing a Longest Common Substring (LCS).

Maximum Matching Model. In [14], the authors introduced an heuristic that aimed at computing a matching between two genomes. This heuristic is a greedy algorithm based on the notion of *LCS*. Let G_0 and G_1 be two genomes: an *LCS* of (G_0, G_1) is a longest common word S of G_0 and G_1, up to a complete reversal. The idea of the greedy algorithm is to match, at each iteration, all the genes that are in an *LCS*. If there are several *LCS*, one is chosen arbitrarily. In [1], we improved this heuristic in the following way: at each iteration, not only we match an *LCS*, but we also remove each unmatched gene of a genome, for which there is no unmatched gene of same family in the other genome. These rules imply that the resulting matching is a maximum matching. We call this heuristic `IILCS_MM`.

Exemplar Model. For the *exemplar* model, we use the same strategy (iteratively match the genes of an *LCS*), except that in this case we must make sure that only *one gene* from each family is matched on each genome. Therefore, at each

iteration, and for each gene **g** present in the *LCS* (and thus kept in the matching), we remove all the other occurrences of **g** in both genomes. Let us call this heuristic `IILCS_EX`.

We have tested both `IILCS_MM` and `IILCS_EX` under, respectively, the *maximum matching* and *exemplar* models. Current results are given in Tables 1 to 4 (see `http://www.lri.fr/~thevenin/Breakpoint/#Some_results` for up-to-date results). The two heuristics are quite fast and one can obtain all results for `IILCS_MM` and `IILC_EX` within 20 minutes on a regular desktop computer. For the *maximum matching* model, Heuristic `IILCS_MM` provides results that are on average 99.11% of the optimal number of breakpoints, ranging from 95.51% to 100%. We actually note that in 14 out of the 66 cases, `IILCS_MM` returns the optimal value. Concerning `IILCS_EX`, the average, obtained over the 49 instances for which we know the optimal result, is 96.88%, ranging from 94.38% to 99.10%.

Genomes	Number of Breakpoints (*maximum matching* model)										
Ecoli	156										
Haein	270	665									
Paeru	240	1082	615								
Pmult	259	703	525	681							
Salty	158	277	676	1091	704						
Wglos	170	194	277	260	270	192					
Xaxon	226	842	533	1016	557	854	269				
Xcamp	226	845	530	1012	555	854	268	181			
Xfast	236	564	468	572	481	569	272	400	404		
Ypest-co92	170	596	649	990	671	591	193	760	755	542	
Ypest-kim	176	607	653	1004	676	606	197	760	749	545	59
	Baphi	Ecoli	Haein	Paeru	Pmult	Salty	Wglos	Xaxon	Xcamp	Xfast	Ypest-co92

Table 1. Exact number of breakpoints for the *maximum matching* model

We thus conclude that both heuristics `IILCS_MM` and `IILCS_EX`, despite being extremely simple and fast, appear to be very good on the dataset we studied. In particular, for the *exemplar* model, since our pseudo-boolean approach seems to reach its limits for some instances, it could be convenient to compute those remaining instances using Heuristic `IILCS_EX`.

Genomes	Number of Breakpoints (*exemplar* model)										
Ecoli	152										
Haein	265	610									
Paeru	232		550								
Pmult	254	622		592							
Salty	154		612		622						
Wglos	168	183	267	248	262	181					
Xaxon	222	675	473		495	684	261				
Xcamp	222	678	473		495		260				
Xfast	231	491	424	499	436	497	264				
Ypest-co92	166		597		597		182	624	620	473	
Ypest-kim	172		598		601		186	624	618	477	
	Baphi	Ecoli	Haein	Paeru	Pmult	Salty	Wglos	Xaxon	Xcamp	Xfast	Ypest-co92

Table 2. Exact number of breakpoints for the *exemplar* model (49 instances out of 66)

Genomes	Number of Breakpoints (*maximum matching* model) for Heuristic `IILCS_MM`										
Ecoli	157										
Haein	270	670									
Paeru	241	1097	619								
Pmult	259	705	529	684							
Salty	158	290	680	1101	708						
Wglos	171	195	277	262	270	193					
Xaxon	226	848	533	1023	560	863	269				
Xcamp	226	851	532	1023	559	860	269	185			
Xfast	236	569	468	575	481	571	272	406	408		
Ypest-co92	173	618	655	1007	678	609	195	767	766	549	
Ypest-kim	178	628	660	1019	684	626	198	766	758	550	59
	Baphi	Ecoli	Haein	Paeru	Pmult	Salty	Wglos	Xaxon	Xcamp	Xfast	Ypest-co92

Table 3. Number of breakpoints for the *maximum matching* model by IILCS_MM

5 Conclusion

In this paper, we presented a method that helps speeding-up computations of exact results for comparing whole genomes containing duplicates. This method,

Genomes	Number of Breakpoints (*exemplar* model) for Heuristic `IILCS_EX`										
Ecoli	155										
Haein	268	636									
Paeru	238	888	571								
Pmult	258	657	509	619							
Salty	156	175	641	908	659						
Wglos	170	189	272	254	266	188					
Xaxon	224	712	494	844	516	722	264				
Xcamp	224	716	492	841	516	720	263	126			
Xfast	234	511	443	517	456	514	268	384	383		
Ypest-co92	171	482	619	829	620	491	188	650	648	490	
Ypest-kim	176	485	624	827	623	492	191	649	644	495	34
	Baphi	Ecoli	Haein	Paeru	Pmult	Salty	Wglos	Xaxon	Xcamp	Xfast	Ypest-co92

Table 4. Number of breakpoints for the *exemplar* model by IILCS_EX

which makes use of pseudo-boolean programming, has been introduced in [1] for computing the maximum number of common intervals between two genomes, and can be used for several (dis)similarity measures. In this paper, we used this method for computing the minimum number of breakpoints between two genomes, and developed pseudo-boolean programs for both the *maximum matching* and *exemplar* models. Experiments were undertaken on a dataset of γ-Proteobacteria, showing the validity of our approach, since all the results (resp. 49 results out of 66) have been obtained in a limited amount of time in the *maximum matching* model (resp. *exemplar* model). Moreover, these results allow us to state that both the `IILCS_MM` and the `IILCS_EX` heuristics provide excellent results on this dataset, hence showing their validity and robustness. On the whole, these preliminary results are very encouraging.

There is still a great amount of work to be done. For instance:

- Implementing and testing the *maximum matching* and the *exemplar* models, for several other (dis)similarity measures,
- For each case, determining strong and relevant rules for speeding-up the process by avoiding the generation of a large number clauses and variables (a pre-processing step that should not be underestimated),
- Obtaining exact results for each of these models and measures, and for different datasets, that could be later used as benchmarks in order to validate (or not) possible heuristics, and
- Implementing and testing an intermediate model between the *maximum matching* and the *exemplar* models, in which one must match *at least* one gene of each family in each genome.

References

1. S. Angibaud, G. Fertin, I. Rusu, and S. Vialette. How pseudo-boolean programming can help genome rearrangement distance computation. In Springer, editor, *Proc. 5th RECOMB Comparative Genomics Satellite Workshop*, volume 4205 of *LNBI*, pages 75–86. springer, 2006.
2. S. Angibaud, G. Fertin, I. Rusu, and S. Vialette. A general framework for computing rearrangement distances between genomes with duplicates. *Journal of Computational Biology*, 14(4):379–393, 2007.
3. P. Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-boolean optimization. Technical Report MPI-I-95-2-003, Max Planck Institut Informatik, 2005. 13 pages.
4. G. Blin, C. Chauve, and G. Fertin. The breakpoint distance for signed sequences. In *Proc. 1st Algorithms and Computational Methods for Biochemical and Evolutionary Networks (CompBioNets)*, pages 3–16. KCL publications, 2004.
5. G. Blin, C. Chauve, and G. Fertin. Genes order and phylogenetic reconstruction: Application to γ-proteobacteria. In *Proc. 3rd RECOMB Comparative Genomics Satellite Workshop*, volume 3678 of *LNBI*, pages 11–20, 2005.
6. G. Blin and R. Rizzi. Conserved intervals distance computation between non-trivial genomes. In *Proc. 11th Annual Int. Conference on Computing and Combinatorics (COCOON)*, volume 3595 of *LNCS*, pages 22–31, 2005.
7. G. Bourque, Y. Yacef, and N. El-Mabrouk. Maximizing synteny blocks to identify ancestral homologs. In *Proc. 3rd RECOMB Comparative Genomics Satellite Workshop*, volume 3678 of *LNBI*, pages 21–35, 2005.
8. D. Bryant. The complexity of calculating exemplar distances. In *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, pages 207–212. Kluwer, 2000.
9. D. Chai and A. Kuehlmann. A fast pseudo-boolean constraint solver. In *Proc. 40th ACM IEEE Conference on Design Automation*, pages 830–835, 2003.
10. C. Chauve, G. Fertin, R. Rizzi, and S. Vialette. Genomes containing duplicates are hard to compare. In *Proc Int. Workshop on Bioinformatics Research and Applications (IWBRA)*, volume 3992 of *LNCS*, pages 783–790, 2006.
11. X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, 2005.
12. N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
13. E. Lerat, V. Daubin, and N.A. Moran. From gene tree to organismal phylogeny in prokaryotes: the case of γ-proteobacteria. *PLoS Biology*, 1(1):101–109, 2003.
14. M. Marron, K.M. Swenson, and B.M.E. Moret. Genomic distances under deletions and insertions. *Theoretical Computer Science*, 325(3):347–360, 2004.
15. D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.
16. D. Sankoff and L. Haque. Power boosts for cluster tests. In *Proc. 3rd RECOMB Comparative Genomics Satellite Workshop*, volume 3678 of *LNBI*, pages 11–20, 2005.
17. A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1998.
18. H.M. Sheini and K.A. Sakallah. Pueblo: A hybrid pseudo-boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:165–189, 2006.