How Pseudo-Boolean Programming can help Genome Rearrangement Distance Computation

Sébastien Angibaud, Guillaume Fertin, Irena Rusu, Stéphane Vialette

How Pseudo-Boolean Programming can help Genome Rearrangement Distance Computation

Sébastien Angibaud[1], Guillaume Fertin[1], Irena Rusu[1], 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
vialette@lri.fr

Abstract. Computing genomic distances between whole genomes is a fundamental problem in comparative genomics. Recent researches have resulted in different genomic distance definitions: number of breakpoints, number of common intervals, number of conserved intervals, Maximum Adjacency Disruption number (MAD), *etc.* Unfortunately, it turns out that, in presence of duplications, most problems are **NP**–hard, and hence several heuristics have been recently proposed. 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. In this paper, we present a novel generic pseudo-boolean approach for computing the exact genomic distance between two whole genomes in presence of duplications, and put strong emphasis on common intervals under the maximum matching model. Of particular importance, we show very strong evidence that the simple LCS heuristic provides very good results on a well-known public benchmark dataset of γ-Proteobacteria.

1 Introduction

Due to the increasing amount of completely sequenced genomes, the comparison of gene order to find conserved gene clusters is becoming a standard approach in comparative genomics. A natural way to compare species is to compare their whole genomes, where comparing two genomes is very often realized by determining a measure of similarity (or dissimilarity) between them.

Several similarity (or dissimilarity) measures between two whole genomes have been recently proposed, such as the number of breakpoints [12,6,2], the number of reversals [6,9], the number of conserved intervals [4], the number of common intervals [5], the Maximum Adjacency Disruption Number (MAD) [13], *etc.* However, in the presence of duplications and for each of the above measures, one has first to disambiguate the data by inferring homologs, *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 [12], for all gene families, all but one occurrence in each genome is deleted. In the maximum matching model [2,8], 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 matching), the problem becomes **NP**–complete as soon as duplicates are present in genomes [6,2,4,8]; 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 [3,5]. 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.

In the present paper, we introduce a novel generic pseudo-boolean programming approach for computing exact solutions. In this first attempt, we focus on the problem of finding the maximum number of common intervals between two genomes under the maximum matching model. For one, from a computational point of view, this problem (together with MAD) is one of the hardest in our pseudo-boolean framework. For another, this allows us to present with a single example the main idea of our approach: a pseudo-boolean program together with reduction rules. Our approach is in fact more ambitious. Our long term goal is indeed to develop a generic pseudo-boolean approach for the exact computation of various genome distances (number of breakpoints, number of common intervals, number of conserved intervals, MAD, *etc.*) under both the exemplar and the maximum matching models, and use this generic approach on different datasets. The rationale of this approach is threefold:

1. There is a crucial need for new algorithmic solutions providing exact genome distances under both the exemplar and the maximum matching model in order to estimate the accuracy of existing heuristics and to design new efficient biologically relevant heuristics.
2. Very little is known about the relations between the various genome distances that have been defined so far (number of breakpoints, number of common intervals, number of conserved intervals, MAD, *etc.*). We thus propose to extensively compared all these genome distances under both models with a generic pseudo-boolean framework on several datasets.
3. We also plan to further investigate the relations between the exemplar and the maximum matching models. We strongly believe here that, in the light of these comparisons, some biologically relevant *intermediate* model between these two extrema could be defined.

This paper is organised as follows. In Section 2, we present some preliminaries and definitions. We focus in Section 3 on the problem of finding the maximum number of common intervals under the maximum matching model and give a pseudo-boolean programming approach together with some reduction rules. Section 4 is devoted to experimental results on a dataset of γ-Proteobacteria. Of particular importance, we show strong evidence that the simple LCS heuristic provides very good results on our dataset.

2 Preliminaries

Genomes with duplications are usually represented by signed sequences over the alphabet of *gene families*, where every element in a genome is a *gene*. However, in order to simplify notations, and since common intervals do not depend on the sign given to the genes, we will consider only *unsigned* genomes. Any gene belongs to a gene family, and two genes belong to the same gene family if they have the same label, regardless of the sign. In the sequel, we will be extensively concerned with pairs of genomes. Let G_1 and G_2 be two genomes, and let $a \in \{0, 1\}$. The number of genes in genome G_a is always written n_a. We denote the i-th gene of genome G_a by $G_a[i]$. For any $1 \leq i \leq j \leq n_a$, we write $\mathcal{G}_a(i, j)$ for the set $\{G_a[i], G_a[i+1], \ldots, G_a[j]\}$ and we let $\mathcal{G}_a$ stand for $\mathcal{G}_a(1, n_a)$. In other words, $\mathcal{G}_a(i, j)$ is the set of all distinct genes between positions i and j in genome G_a, while $\mathcal{G}_a$ is the set of all distinct genes in the whole genome G_a. For any gene $\mathbf{g} \in \mathcal{G}_a$ and any $1 \leq i \leq j \leq n_a$, we denote by $\mathsf{occ}_a(\mathbf{g}, i, j)$ the number of occurrences of gene $\mathbf{g}$ in the sequence $(G_a[i], G_a[i+1], \ldots, G_a[j])$. To simplify notations, we abbreviate $\mathsf{occ}_a(\mathbf{g}, 1, n_a)$ to $\mathsf{occ}_a(\mathbf{g})$.

A *matching* $\mathcal{M}$ between genomes G_1 and G_2 is a set of pairwise disjoint pairs $(G_1[i], G_2[j])$, where $G_1[i]$ and $G_2[j]$ belong to the same gene family, *i.e.*, $G_1[i] = G_2[j]$. Genes of G_1 and G_2 that do not belong to any pair of the matching $\mathcal{M}$ are said to be *unmatched* for $\mathcal{M}$. A matching $\mathcal{M}$ between G_1 and G_2 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_1 and G_2, respectively. A matching $\mathcal{M}$ between G_1 and G_2 can be seen as a way to describe a putative assignment of orthologous pairs of genes between G_1 and G_2 (see for example [9]).

Let $\mathcal{M}$ be any matching between G_1 and G_2. By first deleting unmatched genes and next renaming genes in G_1 and G_2 according to the matching $\mathcal{M}$, we may now assume that both G_1 and G_2 are duplication free, *i.e.*, G_2 is a permutation of G_1. A *common interval* between G_1 and G_2 is a substring of G_1, *i.e.*, a sequence of consecutive genes of G_1, for which the exact same content can be found in a substring of G_2. It is easily seen that, by first resorting to a renaming procedure, we can always assume that one of the two genomes, say G_1, is the identity permutation, *i.e.*, $G_1 = 1\ 2\ \ldots\ n_1$. For example, let $G = 1\ 2\ 3\ 4\ 5$ and $G_2 = 1\ 5\ 3\ 4\ 2$ then the interval $[3 : 5]$ of G_1 is a common interval (because 5 3 4 occurs as a substring in G_2). Notice that there exist at least $n + 1$ ($n = n_1 = n_2$) common intervals between G_1 and G_2 since each individual gene is always a common interval and G_1 itself is also a common interval. This lower bound is tight as shown by $G_1 = 1\ 2\ 3\ 4$ and $G_2 = 2\ 4\ 1\ 3$. Furthermore, if $G_1 = G_2$ the number of common intervals between G_1 and G_2 is $\frac{n(n+1)}{2}$, where $n = n_1 = n_2$, *i.e.*, each possible substring of G_1 is a common interval.

3 An exact algorithm for maximising the number of common intervals

3.1 Pseudo-boolean models

A Linear Pseudo-boolean (LPB) program is a linear program [14] where all variables are restricted to take values of either 0 or 1. For one, LPB programs are viewed by the linear programming community as just a domain restriction on general linear programming. For another, from a satisfiability (Sat) point of view, pseudo-boolean constraints can be seen as a *generalisation* of clauses providing a significant extension of purely propositional constraints [7,10].

Conventionally, LPB problems are handled by generic Integer Linear Programming (ILP) solvers. The drawback of such an approach is that generic ILP solvers typically ignore the boolean nature of the variables. Alternatively, LPB decision problems could be encoded as Sat instances in pure CNF (Conjunctive Normal Form), *i.e.*, conjunction of disjunctions of boolean literals, which are then solved by any of the highly specialised Sat approaches. However the number of clauses required for expressing the LPB constraints is prohibitively large. Moreover a pure CNF encoding may prevent the solver from pruning the search space effectively [7]. Boolean satisfiability solvers available today are the result of decades of research and are deemed to be among the faster **NP**–complete problem specific solvers. The latest generation of Sat solver generally have three key features (randomisation of variable selection, backtracking search and some form of clause learning) and usually run in reasonable time (even for very large instances).

A number of generalisations of Sat solvers to LPB solvers have been proposed (`Pueblo` [15], `Galena` [7], `OPBDP` [1] and more). We decided to use for our tests the `minisat+` LPB solver [10] because of its good results during PB evaluation 2005 (special track of the Sat Competition 2005).

3.2 Common intervals

We propose in Figure 1 a pseudo-boolean program for computing the maximum number of common intervals between two genomes under the maximum matching model in the presence of duplications (we assume here that each gene $\mathbf{g} \in \mathcal{G}_1 \cup \mathcal{G}_2$ occurs both in G_1 and in G_2).

Program `Common-Intervals-Matching` is clearly a pseudo-boolean program, *i.e.*, a $(0,1)$-linear program. Roughly speaking, the boolean variables are divided in two sets: true setting of variables in C denote possible common intervals between G_1 and G_2, while true setting of variables in X denote the mapping, *i.e.*, matching, between G_1 and G_2. We now turn to describing the constraints. Constraints in (`C.01`) and in (`C.02`) deal with consistency of the mapping: each gene of G_1 is mapped to at most one gene of G_2, and conversely (some genes need indeed to be deleted in case of unbalanced families). Constraints in (`C.03`) ensure that each common interval is counted exactly once. The key idea here is to impose an "*active border*" property, *i.e.*, if variable $c^{i,j}_{k,\ell}$ is set to 1 then genes

Program `Common-Intervals-Matching`

objective:

$$\text{maximize} \sum_{c_{k,\ell}^{i,j} \in A} c_{k,\ell}^{i,j}$$

variables:

$$C = \{c_{k,\ell}^{i,j} : 1 \leq i \leq j \leq n_1 \wedge 1 \leq k \leq \ell \leq n_2\}$$
$$X = \{x_k^i : 1 \leq i \leq n_1 \wedge 1 \leq k \leq n_2 \wedge G_1[i] = G_2[k]\}$$

subject to:

$$(\texttt{C.01}) \quad \forall i = 1, 2, \ldots, n_1, \quad \sum_{\substack{1 \leq k \leq n_2 \\ G_1[i] = G_2[k]}} x_k^i \leq 1$$
$$(\texttt{C.02}) \quad \forall k = 1, 2, \ldots, n_2, \quad \sum_{\substack{1 \leq i \leq n_1 \\ G_1[i] = G_2[k]}} x_k^i \leq 1$$
$$(\texttt{C.03}) \quad \forall c_{k,\ell}^{i,j} \in C, \quad 4\, c_{k,\ell}^{i,j} - \sum_{\substack{k \leq r \leq \ell \\ G_1[i] = G_2[r]}} x_r^i - \sum_{\substack{k \leq s \leq \ell \\ G_1[j] = G_2[s]}} x_s^j - \sum_{\substack{i \leq p \leq j \\ G_1[p] = G_2[k]}} x_k^p - \sum_{\substack{i \leq q \leq j \\ G_1[q] = G_2[\ell]}} x_\ell^q \leq 0$$
$$(\texttt{C.04}) \quad \forall c_{k,\ell}^{i,j} \in C,\ \forall i < p < j,\ \forall 1 \leq r < k, \quad G_1[p] = G_2[r], \quad c_{k,\ell}^{i,j} + x_r^p \leq 1$$
$$(\texttt{C.05}) \quad \forall c_{k,\ell}^{i,j} \in C,\ \forall i < p < j,\ \forall \ell < r \leq n_2,\ G_1[p] = G_2[r], \quad c_{k,\ell}^{i,j} + x_r^p \leq 1$$
$$(\texttt{C.06}) \quad \forall c_{k,\ell}^{i,j} \in C,\ \forall k < r < \ell,\ \forall 1 \leq p < i, \quad G_1[p] = G_2[r], \quad c_{k,\ell}^{i,j} + x_r^p \leq 1$$
$$(\texttt{C.07}) \quad \forall c_{k,\ell}^{i,j} \in C,\ \forall k < r < \ell,\ \forall j < p \leq n_1,\ G_1[p] = G_2[r], \quad c_{k,\ell}^{i,j} + x_r^p \leq 1$$
$$(\texttt{C.08}) \quad \forall \mathbf{g} \in \mathcal{G}_1 \cup \mathcal{G}_2, \quad \sum_{\substack{1 \leq i \leq n_1 \\ G_1[i] = \mathbf{g}}} \sum_{\substack{1 \leq k \leq n_2 \\ G_2[k] = \mathbf{g}}} x_k^i = \min\{\mathsf{occ}_1(\mathbf{g}), \mathsf{occ}_2(\mathbf{g})\}$$

domains:

$$\forall x_k^i \in X, \quad x_k^i \in \{0, 1\}$$
$$\forall c_{k,\ell}^{i,j} \in C, \quad c_{k,\ell}^{i,j} \in \{0, 1\}$$

Fig. 1. Program `Common-Intervals-Matching` for finding the maximum number of common intervals between two genomes under the maximum matching model.

$G_1[i]$ and $G_1[j]$ must match some distinct genes between positions k and ℓ in G_2, and genes $G_2[k]$ and $G_2[\ell]$ must match some distinct genes between positions i and j in G_1. Constraints in (`C.04`) to (`C.07`) ensure that if $c_{k,\ell}^{i,j} = 1$ then the interval $[i : j]$ of G_1 and the interval $[k : \ell]$ of G_2 is a common interval according to the mapping induced by the true setting of X. For example, constraints in (`C.04`) ensure that each gene in the interval $[i : j]$ of G_1 is either not mapped or is mapped to a gene in the interval $[k : \ell]$ of G_2 (thanks to constraints in (`C.01`), (`C.02`) and (`C.03`), genes at position i and j in G_1 are actually mapped to distinct genes in G_2 if $i < j$ and $c_{k\ell}^{i,j} = 1$). Finally, constraints in (`C.08`) forces the mapping to be a maximum matching between G_1 and G_2.

Proposition 1. *Program* `Common-Intervals-Matching` *correctly computes the maximum number of common intervals between G_1 and G_2 under the maximum matching model.*

We briefly discuss here space issues of Program `Common-Intervals-Matching`. First, it is easily seen that $\#C = \Theta(n_1^2\, n_2^2)$ and hence that (`C.03`) is composed of $\Theta(n_1^2\, n_2^2)$ constraints. The number of constraints in (`C.04`) to (`C.07`) however does depend on the number of duplications in the two genomes. Second, $\#X = \mathcal{O}(n_1\, n_2)$. Clearly, the size of the set X determines the number of constraints in (`C.01`) and (`C.02`) and of course strongly depends on the number of duplications in G_1 and G_2. Not surprisingly, the set X turns out to be of moderate size in practice. Finally, the number of constraints in (`C.07`) is clearly linear in the size of the two genomes. We shall soon describe (section 3.3) how to speed-up the program by reducing the number of variables and constraints.

We observe that replacing constraints in (`C.08`) by a new set of constraints (`C.08'`) – see below – in Program `Common-Intervals-Matching` results in the pseudo-boolean program `Common-Intervals-Exemplar` that computes the maximum number of common intervals between genomes G_1 and G_2 under the exemplar model.

$$\texttt{(C.08')}\quad \forall \mathbf{g} \in \mathcal{G}_1 \cup \mathcal{G}_2, \qquad \sum_{\substack{1 \le i \le n_1 \\ G_1[i] = \mathbf{g}}} \; \sum_{\substack{1 \le k \le n_2 \\ G_2[k] = \mathbf{g}}} x_k^i = 1$$

Interestingly enough, substituting now the constraints in (`C.08`) by a new set of constraints (`C.08''`) – see below – in Program `Common-Intervals-Matching` results in a pseudo-boolean program that computes the maximum number of common intervals between genomes G_1 and G_2 under the following intermediate model: at least one gene in each gene family is mapped. Observe that this model contains both the exemplar model and the maximum matching model as special cases.

$$\texttt{(C.08'')}\quad \forall \mathbf{g} \in \mathcal{G}_1 \cup \mathcal{G}_2, \qquad \sum_{\substack{1 \le i \le n_1 \\ G_1[i] = \mathbf{g}}} \; \sum_{\substack{1 \le k \le n_2 \\ G_2[k] = \mathbf{g}}} x_k^i \ge 1$$

3.3 Speeding-up the program

We give in this section four rules for speeding-up Program `Common-Intervals-Matching`. In theory, a very large instance may be easy to solve and a small instance hard. However, very often, small hard instances turn out be artificial, *e.g.*, the pigeonhole problem, and hence, in case of practical instances, the running time of a pseudo-boolean solver is most of the time related to the size of the instances. The main idea here is thus to reduce the number of variables and constraints in the program (for ease of exposition we describe our rules as filters on C). More precisely, we give rules that avoid introducing useless variables $c_{k,\ell}^{i,j}$ in

C in such a way that the correctness of Program `Common-Intervals-Matching` is maintained by repeated applications of the rules; two of these filters however do modify the correct maximum number of common intervals between the two genomes and thus ask for subsequent modifications in order to obtain the correct solution.

[**Rule 1**] Delete from C all variables $c_{k,k}^{i,i}$, $1 \leq i \leq n_1$ and $1 \leq k \leq n_2$.

[**Rule 1**] does modify the correct number of common intervals between G_1 and G_2, and hence application of this rule asks for subsequent modifications of the number of common intervals. The key idea of [**Rule 1**] is simply to discard common intervals of size 1 from the program. Indeed, we can compute in a preprocessing step the numbers d_1 and d_2 of genes that need to be deleted in G_1 and G_2 for obtaining a maximum matching between the two genomes. Therefore, we know that the resulting genomes will consist in $L = n_1 - d_1 = n_2 - d_2$ genes, where

$$L = \sum_{\mathbf{g} \in \mathcal{G}} \min\{\mathsf{occ}_1(\mathbf{g}), \mathsf{occ}_2(\mathbf{g})\}.$$

But each of these genes will contribute for 1 to the number of common intervals between G_1 and G_2, for any maximum matching. We thus simply delete all these variables and add L to the number of common intervals between G_1 and G_2 found by Program `Common-Intervals-Matching`.

[**Rule 2**] Delete from C all variables $c_{k,\ell}^{i,j}$ for which any of the following conditions holds true:

1. $(\#\{r : k \leq r \leq \ell \wedge G_1[i] = G_2[r]\} = 0) \vee (\#\{s : k \leq s \leq \ell \wedge G_1[j] = G_2[s]\} = 0)$,
2. $(\#\{r : k \leq r \leq \ell \wedge G_1[i] = G_2[r]\} < 2) \wedge (G_1[i] = G_1[j])$,
3. $(\#\{p : i \leq p \leq j \wedge G_2[k] = G_1[p]\} = 0) \vee (\#\{q : i \leq q \leq j \wedge G_2[\ell] = G_1[q]\} = 0)$,
4. $(\#\{p : i \leq p \leq j \wedge G_2[k] = G_1[p]\} < 2) \wedge (G_2[k] = G_2[\ell])$.

[**Rule 2**] is a quickening for constraints in (`C.03`). Indeed, these constraints ensure that each common interval is counted exactly once by the program by forcing the border of each common interval to be active in the computed solution, *i.e.*, genes $G_1[i]$ and $G_1[j]$ match some distinct genes between positions k and ℓ in G_2, and genes $G_2[k]$ and $G_2[\ell]$ match some distinct genes between positions i and j in G_1. Correctness of [**Rule 2**] thus follows from the fact that Program `Common-Intervals-Matching` will always set a variable $c_{k,\ell}^{i,j}$ to 0 if the border property cannot be satisfied (it is assumed here that $i < j$ and $k < \ell$).

[**Rule 3**] Delete from C all variables $c_{k,\ell}^{i,j}$ for which there exists at least one gene $\mathbf{g} \in \mathcal{G}$ such that $|\mathsf{occ}_1(\mathbf{g}, i, j) - \mathsf{occ}_2(\mathbf{g}, k, \ell)| > |\mathsf{occ}_1(\mathbf{g}) - \mathsf{occ}_2(\mathbf{g})|$.

Roughly speaking, [**Rule 3**] avoids us to kill too many genes in a common interval. Indeed, for one, for any $\mathbf{g} \in \mathcal{G}$, $|\mathsf{occ}_1(\mathbf{g}, i, j) - \mathsf{occ}_2(\mathbf{g}, k, \ell)|$ is clearly the minimum number of occurrences of gene $\mathbf{g}$ that need to be deleted if $c_{k,\ell}^{i,j} = 1$, *i.e.*, $[i : j]$ and $[k : \ell]$ form a common interval between the two genomes. For another, for any $\mathbf{g} \in \mathcal{G}$, $|\mathsf{occ}_1(\mathbf{g}) - \mathsf{occ}_2(\mathbf{g})|$ is the number of occurrences of gene $\mathbf{g}$ that need to be deleted in G_1 and G_2 for finding any maximum matching between the two genomes. Correctness of [**Rule 3**] thus follows from the fact that we can certainly not delete more than $|\mathsf{occ}_1(\mathbf{g}) - \mathsf{occ}_2(\mathbf{g})|$ occurrences of gene $\mathbf{g}$.

[**Rule 4**] Delete from C all variables $c^{i,j}_{k,\ell}$ for which the four following conditions hold true:

1. $\forall \mathbf{g} \in \mathcal{G}_1(i,j), \quad \#\mathsf{occ}_1(\mathbf{g}, 1, i-1) + \#\mathsf{occ}_1(\mathbf{g}, j+1, n_1) = 0,$
2. $\forall \mathbf{g} \in \mathcal{G}_2(k,\ell), \quad \#\mathsf{occ}_2(\mathbf{g}, 1, k-1) + \#\mathsf{occ}_2(\mathbf{g}, \ell+1, n_2) = 0,$
3. $\#\mathsf{occ}_1(G_1[i]) \leq \#\mathsf{occ}_2(G_1[i])$ and $\#\mathsf{occ}_1(G_1[j]) \leq \#\mathsf{occ}_2(G_1[j]),$
4. $\#\mathsf{occ}_2(G_2[k]) \leq \#\mathsf{occ}_1(G_2[k])$ and $\#\mathsf{occ}_2(G_2[\ell]) \leq \#\mathsf{occ}_1(G_2[\ell]).$

We first observe that [**Rule 4**] does modify the correct number of common intervals between G_1 and G_2, and hence application of this rule asks for subsequent modifications of the number of common intervals. The rationale of [**Rule 4**] is that, if the four conditions hold true, then $c^{i,j}_{k,\ell}$ will always be set to 1 by Program `Common-Intervals-Matching`. In other words, for any maximum matching between G_1 and G_2, $[i : j]$ and $[k : \ell]$ will form a common intervals. We thus simply delete from C all these variables $c^{i,j}_{k,\ell}$ and add the number of deleted variables by [**Rule 4**] to the number of common intervals between G_1 and G_2 found by Program `Common-Intervals-Matching`. This rule will prove extremely useful for highly conserved regions with localised duplications.

4 Experimental results

As mentioned earlier, the generic pseudo-boolean approach we propose in this paper can be useful for estimating the accuracy of an heuristic. In that sense, it is necessary to compute the exact results for different datasets, that could be later used as benchmarks to which confront any given heuristic algorithm.

Computing exact results for different datasets and different (dis)similarity measures is a long task, because the problem is **NP**–hard (see for instance [6,2,8]), which implies that there is no guarantee that a computer (even a very powerful one) will ever provide an exact result ; however, the main interest of the pseudo-boolean approach is that, due to several decades of research intended in speeding-up the computation process, this specific problem can be solved in reasonable time, even for very large instances.

We started the computation of exact results concerning common intervals in the maximum matching model by studying the dataset used in [3]. This dataset is composed of 12 complete genomes from the 13 γ-Proteobacteria originally studied in [11]. The thirteenth genome (*V.cholerae*) has not been considered, since it is composed of two chromosomes, and hence does not fit in the model we considered here for representing genomes. More precisely, this dataset is composed of the genomes of the following species: *Buchnera aphidicola APS* (Genbank accession number NC_002528), *Escherichia coli K12* (NC_000913), *Haemophilus* *influenzae Rd* (NC_000907), *Pasteurella multocida Pm70* (NC_002663), *Pseudomonas aeruginosa PA01* (NC_002516), *Salmonella typhimurium LT2* (NC_003197), *Xanthomonas axonopodis pv. citri 306* (NC_003919), *Xanthomonas campestris* (NC_0 03902), *Xylella fastidiosa 9a5c* (NC_002488), *Yersinia pestis CO_92* (NC_003143), *Yersinia pestis KIM5 P12* (NC_004088), *Wigglesworthia glossinidia brevipalpis* (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 [3].

The results we have obtained are given in Table 1. Out of the 66 possible pairwise genome comparisons, 38 results have been obtained so far. Despite the fact that the variant and the measure we study is one of the most time consuming (as mentioned in Section 1), the results look promising, since more than half of the results have been computed until now. Moreover, among those 38 values, only 2 of them took several days to be computed, while the others took no more than a few minutes.

	Baphi	Ecoli	Haein	Paeru	Pmult	Salty	Wglos	Xanon	Xcamp	Xfast	Ypest-co92
Ecoli	2882										
Haein	1109	2784									
Paeru	1524		2036								
Pmult	1224	3342	3936								
Salty	2849		2820		3376						
Wglos	1275	2328	1085	1558	1214	2335					
Xanon	1183		1471				1225				
Xcamp	1183		1458				1223				
Xfast	979	1877	1295			1981	994				
Ypest-co92	2585		2694		3298		2318			1949	
Ypest-kim	2141		2500		3092	2093				1891	

Table 1. Number of common intervals in the maximum matching model: exact results obtained by our pseudo-boolean transformation (38 out of 66)

In addition to being promising because they were obtained in a reasonable amount of time, these results, though still partial, allow us to go further. Indeed, they will allow us (i) to discuss the heuristic used in [3] (that we will denote *ILCS*), (ii) to discuss an improvement we suggest for *ILCS* (that we will denote *IILCS*), and (iii) to compare the results of *IILCS* to the exact results we have obtained via our pseudo-boolean approach. We first start by describing the heuristic used in [3], that we will call *ILCS* (Iterative Longest Common Substring). This heuristic is greedy, and works as follows:

1. Compute the Longest Common Substring (*i.e.*, the longest contiguous word) S of the two genomes, up to a complete reversal. If there are several candidates, pick one arbitrarily
2. Match all the genes of S accordingly
3. Iterate the process until all possible genes have been matched (*i.e.*, we have obtained a maximum matching)
4. Remove, in each genome, all the genes that have not been matched

5. Compute the number of common intervals that have been obtained in this solution

The simple idea behind this heuristic algorithm is that an LCS (up to complete reversal) of length k contains $\frac{k(k+1)}{2}$ common intervals. Hence, finding such exact copies in both genomes intuitively helps to increase the total number of common intervals. The results obtained by the heuristic *ILCS* in [3] are summarised in Table 2. Since the results are not symmetric (*i.e.*, running *ILCS*(G_1, G_2) on genomes G_1 and G_2 does not necessarily produce the same number of common intervals than running *ILCS*(G_2, G_1) on the same genomes taken in the other order), we took, for each genome comparison, the best result.

	Baphi	Ecoli	Haein	Paeru	Pmult	Salty	Wglos	Xanon	Xcamp	Xfast	Ypest-co92
Ecoli	2605										
Haein	1104	2758									
Paeru	1494	3862	1981								
Pmult	1219	3297	3901	2278							
Salty	2641	65634	2794	3826	3327						
Wglos	1267	2102	1078	1496	1204	2083					
Xanon	1147	2485	1446	3788	1626	2613	1175				
Xcamp	1153	2459	1441	3746	1603	2603	1168	106681			
Xfast	975	1828	1285	2332	1457	1956	963	6797	6647		
Ypest-co92	2414	13818	2668	3887	3237	15007	2037	2395	2358	1810	
Ypest-kim	1943	13762	2460	3757	3027	14770	1846	2471	2446	1854	242963

Table 2. Number of common intervals in the maximum matching model: results obtained by the *ILCS* heuristic from [3]

A deeper study of the *ILCS* heuristic led us to suggest an improvement to it, in the form of a new heuristic, that we call *IILCS* (Improved Iterative Longest Common Substring). The only difference between *IILCS* and *ILCS* is that, before searching for a Longest Common Substring (up to a complete reversal), we "*tidy*" the two genomes, in the sense that we remove, in each genome and at each iteration, the genes for which we know they will not be matched in the final solution (this is simply done by counting, at each iteration, the number of unmatched genes of each gene family). This actually allows to possibly find longer Longest Common Substrings at each iteration, and always gives better results on the studied dataset (except in one case where the result is the same for both heuristics, see Table 3). On average, over the 66 pairwise genome comparisons, *IILCS* improves by 2.6% the number of common intervals that are found. The results for *IILCS* are summarised in Table 3. Similarly to *ILCS*, the results are not symmetric, thus we took, for each genome comparison, the best result.

The most interesting and surprising result, which we were able to point thanks to our pseudo-boolean transformation and the exact results obtained from it, is that heuristic *IILCS* appears to be *very* good on the dataset we studied. Indeed, out of the 38 instances for which we have computed the exact results, *IILCS* returns the *optimal result* in 7 cases, and returns a number of

	Baphi	Ecoli	Haein	Paeru	Pmult	Salty	Wglos	Xanon	Xcamp	Xfast	Ypest-co92
Ecoli	2869										
Haein	1109	2775									
Paeru	1518	3976	2018								
Pmult	1224	3329	3887	2321							
Salty	2849	66025	2809	3956	3350						
Wglos	1267	2186	1085	1508	1211	2274					
Xanon	1183	2540	1468	3952	1644	2685	1198				
Xcamp	1183	2524	1455	3898	1621	2675	1196	108347			
Xfast	979	1849	1293	2365	1464	1974	973	6890	6732		
Ypest-co92	2541	14364	2686	3989	3268	15192	2307	2482	2433	1816	
Ypest-kim	2124	14126	2487	3859	3037	15451	2091	2557	2509	1863	261345

Table 3. Number of common intervals in the maximum matching model: results obtained by the *IILCS* heuristic

common intervals that is more than 99% of the optimal number for 18 other cases. The "worse" result that *IILCS* provides is 93.17% away from the optimal (Ypest-co92/Xfast). On average, over the 38 pairwise comparisons for which we have exact results, *IILCS* performs very well, since it gives a number of common intervals that is 98.91% of the optimal number.

This result comes as a surprise, because, despite being extremely simple and fast, *IILCS* appears to be very good on this dataset. Hence, this strongly suggests that computing common intervals in the maximum matching model can simply be undertaken using *IILCS* while remaining accurate, thus validating this heuristic.

5 Conclusion

In this paper, we have introduced a novel and original method that helps speeding-up computations of exact results for comparing whole genomes containing duplicates. This method makes use of pseudo-boolean programming. Our approach is very general, and can handle several (dis)similarity measures (breakpoints, common intervals, conserved intervals, MAD, *etc.*) under several possible models (exemplar model, maximum matching model, but also most variants within those two extrema). An example of such an approach (common intervals under the maximum matching model) has been developed, in order to illustrate the main ideas of the pseudo-boolean transformation framework that we suggest. Experiments have also been undertaken on a dataset of γ-Proteobacteria, showing the validity of our approach, since already 38 results (out of 66) have been obtained in a limited amount of time. Moreover, those preliminary results have allowed us to state that the new *IILCS* heuristic provides excellent results on this dataset, hence showing its validity and robustness. On the whole, those preliminary results are very encouraging.

There is still a great amount of work to be done, and some of it is being undertaken by the authors at the moment. Among other things, one can cite:

- Implementing and testing all the possible above mentioned variants, for all the possible above mentioned (dis)similarity measures,
- For each case, determine strong and relevant rules for speeding-up the process by avoiding the generation of too many clauses and variables (a preprocessing step that should not be underestimated),
- Obtaining exact results for each of those variants and measures, for different datasets, that could be later used as benchmarks for validating (or not) possible heuristics, but also the measures themselves, or even the models.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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. 2000.
7. D. Chai and A. Kuehlmann. A fast pseudo-boolean constraint solver. In *Proc. 40th ACM IEEE conference on Design automation*, pages 830–835, 2003.
8. 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.
9. 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.
10. N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
11. 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.
12. D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.
13. 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.
14. A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1998.
15. 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.