Pseudo-Boolean Programming for Partially Ordered Genomes

Sébastien Angibaud, Guillaume Fertin, Annelyse Thevenin, Stéphane Vialette

Pseudo Boolean Programming for Partially Ordered Genomes

Sébastien Angibaud[1], Guillaume Fertin[1], Annelyse Thévenin[2] and Stéphane Vialette[3]

[1] Laboratoire d'Informatique de Nantes-Atlantique (LINA), UMR CNRS 6241
Université de Nantes, 2 rue de la Houssinière, 44322 Nantes Cedex 3 - France
[2] Laboratoire de Recherche en Informatique (LRI), UMR CNRS 8623
Université Paris-Sud, 91405 Orsay - France
[3] IGM-LabInfo, UMR CNRS 8049, Université Paris-Est,
5 Bd Descartes 77454 Marne-la-Vallée, France
{Sebastien.Angibaud,Guillaume.Fertin}@univ-nantes.fr, thevenin@lri.fr,
vialette@univ-mlv.fr

Abstract. Comparing genomes of different species is a crucial problem in comparative genomics. Different measures have been proposed to compare two genomes: number of common intervals, number of adjacencies, number of reversals, etc. These measures are classically used between two totally ordered genomes. However, genetic mapping techniques often give rise to different maps with some unordered genes. Starting from a partial order between genes of a genome, one method to find a total order consists in optimizing a given measure between a linear extension of this partial order and a given total order of a close and well-known genome. However, for most common measures, the problem turns out to be **NP**-hard. In this paper, we propose a $(0, 1)$-linear programming approach to compute a linear extension of one genome that maximizes the number of common intervals (resp. the number of adjacencies) between this linear extension and a given total order. Next, we propose an algorithm to find linear extensions of two partial orders that maximize the number of adjacencies.

1 Introduction

Genetic mapping techniques often give rise to different maps with some unordered genes. In that case, maps are combined in the form of a partially ordered sequence of genes, and thus a genome is modeled by a *poset* (i.e., a partially ordered set), as was done in [10]. In this model, any linear extension of a poset represents a possible total order of the genome. In order to find a total order from a partial order, Sankoff et al. [10] suggested the following method: confront the partial order with a given total order of a close and well-known genome. More precisely, this method asks for a linear extension of the partially ordered genome that optimizes a (dis)similarity measure between this total order and a given totally ordered genome. Several measures can be used: number of common intervals [8], of adjacencies [1], of breakpoints [9], of conserved intervals [3], etc.

Computing these measures between two total orders, whenever no gene is duplicated, is a polynomial-time solvable problem. Unfortunately, concerning the number of adjacencies, the number of common intervals and the reversal distance, finding a linear extension that optimizes one of these measures between this extension and a given total order is **NP**-hard [4,6]. In this paper, we first present an approach to compute a linear extension of a partially ordered genome with respect to a given totally ordered genome without duplicated genes. Our method lies on a transformation of the initial problem into a $(0,1)$-linear program (i.e., a linear program with boolean variables) [2,1]. We focus here on two similarity measures: the *number of common intervals* and the *number of adjacencies*. We also extend our approach and consider the problem of comparing *two* partially ordered genomes with respect to the number of adjacencies. After presenting our three algorithms, we evaluate our method by using the same simulated data as in [4].

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 a total order of a partially ordered genome that maximizes either the number of common intervals or the number of adjacencies with a given totally ordered genome. Next, we extend the case of adjacencies to find two totally ordered genomes from two partially ordered genomes. For each problem, we give its formulation in terms of a $(0,1)$-linear program, together with some reduction rules that ail at speeding-up the process. Section 4 is devoted to experimental results on simulated data.

2 Preliminaries

In the literature, a (totally ordered) duplication-free genome having n oriented genes is usually represented by a signed permutation, one where each element in the set $\{1, 2, \ldots, n-1, n\}$ has either a sign, $+$ or $-$. Let T_1 and T_2 be two duplication-free genomes of size n. Wlog, we may assume that T_2 is the identity positive permutation, i.e., $T_2 = +1\ +2\ \ldots\ +n$. An interval of T_1 is a set of m consecutive genes $\{\mathtt{g}_1, \mathtt{g}_2, \ldots, \mathtt{g}_m\}$ in T_1. A common interval between T_1 and T_2 is an interval which occurs in T_1 and in T_2. Observe that the notion of common interval does not take into account the sign of the genes. We say that there is a *breakpoint* after gene $T_1[i]$, $1 \leq i \leq n-1$, in T_1 if neither $T_1[i]$ and $T_1[i+1]$ nor $-T_1[i+1]$ and $-T_1[i]$ are consecutive genes in T_2, otherwise we say that there is an *adjacency* after gene $T_1[i]$. Notice, that in order to take into account common intervals and adjacencies that may occur at the extremities of a genome, we artificially add to a genome with n genes the gene $+0$ (resp. $+(n+1)$) to its left (resp. right). We now introduce the required material for partially ordered genomes. A *partial order* on a set P is a binary relation $\preceq$ such that, for x, y and z in P, (i) $x \preceq x$, (ii) $x \preceq y$ and $y \preceq x$ imply $x = y$, and (iii) $x \preceq y$ and $y \preceq z$ imply $x \preceq z$. These three relations are referred to as *reflexivity*, *antisymmetry*, and *transitivity*, respectively. A set P equipped with a partial order relation is said to be a *partially ordered set* (also called a *poset*).

When it is necessary to specify the partial order relation, we write $(P, \preceq)$. A partial order relation $\preceq$ on P gives rise to a relation $\prec$ of strict inequality: $x \prec y$ in P iff $x \preceq y$ and $x \neq y$. Strict partial orders differ from partial orders only in whether each element is required to be unrelated, or required to be related, to itself. A totally ordered set (or linearly ordered set) is a poset P which has the property that every two elements of P are comparable (i.e., for all $x, y \in P$ either $x \preceq y$ or $y \preceq x$). A poset Q is called an *extension* of a poset P if the elements of P and Q are the same, and the set of relations of P is a subset of the set of relations of Q (i.e., for all $x, y \in P$, if $x \prec y$ in P, then $x \prec y$ in Q, but not necessarily conversely). Q is called a *linear extension* of P if Q is an extension of P and also a linear order. In our context, a totally (resp. partially) ordered genome is thus a set P that contains exactly one occurrence of i or $-i$, $0 \leq i \leq n+1$, equipped with some total (resp. partial) order. For the sake of brevity, the totally ordered genome $+0\ +1\ +2\ \ldots\ +n+(n+1)$ is abbreviated as Id. Let P_1 and P_2 be two partially ordered genomes, and let $x \in \{1, 2\}$. The sign of a gene $\mathtt{g}$ in P_x is written $s_x(\mathtt{g})$. We write $\mathtt{g}_1 \prec_x \mathtt{g}_2$ if gene $\mathtt{g}_1$ precedes gene $\mathtt{g}_2$ in P_x. The number of genes $\mathtt{g}_1$ such that $\mathtt{g}_1 \prec_x \mathtt{g}_2$ is denoted by $\mathrm{prec}_x(\mathtt{g}_2)$. We write $\mathtt{g}_2 \succ_x \mathtt{g}_1$ if gene $\mathtt{g}_2$ follows $\mathtt{g}_1$ in P_x. The number of genes $\mathtt{g}_2$ such that $\mathtt{g}_1 \prec_x \mathtt{g}_2$ is denoted by $\mathrm{succ}_x(\mathtt{g}_1)$. In P_x, two genes $\mathtt{g}_1$ and $\mathtt{g}_2$ are said to be *incomparable* (written $\mathtt{g}_1 \|_x \mathtt{g}_2$) if neither $\mathtt{g}_1 \prec_x \mathtt{g}_2$ nor $\mathtt{g}_2 \prec_x \mathtt{g}_1$. The *width* of a partially ordered genome is the size of a largest subset of incomparable genes. Two genes $\mathtt{g}_1$ and $\mathtt{g}_2$ are said to be *adjoining* in P_x if there does not exist $\mathtt{g}_3$ such that $\mathtt{g}_1 \prec_x \mathtt{g}_3 \prec_x \mathtt{g}_2$ or $\mathtt{g}_2 \prec_x \mathtt{g}_3 \prec_x \mathtt{g}_1$ or $\mathtt{g}_1 \|_x \mathtt{g}_3$ or $\mathtt{g}_2 \|_x \mathtt{g}_3$. Let T_x be a linear extension of P_x. The set of allowed positions of a gene $\mathtt{g}$ in T_x is written by $POS_x(\mathtt{g})$; one can easily check that $POS_x(\mathtt{g}) = \{\mathrm{prec}_x(\mathtt{g})+1, \ldots, \mathrm{succ}_x(\mathtt{g})-1\}$. The position of $\mathtt{g}$ in T_x is written $T_x(\mathtt{g})$ (by definition, $T_x(\mathtt{g}) \in POS_x(\mathtt{g})$). We say that a gene $\mathtt{g}_1$ is *i-nail*$_x$ if $POS_x(\mathtt{g}_1) = \{i\}$ (i.e., there are $i-1$ genes $\mathtt{g}_2$ such that $\mathtt{g}_2 \prec_x \mathtt{g}_1$ and no gene is incomparable with $\mathtt{g}_1$ in P_x). In this paper, we are interested in three combinatorial problems. The first two problems are concerned with confronting a partially ordered genome with a reference totally ordered genome: given a partially ordered genome P_1 and a reference totally ordered genome T_2, problem `MCIL-1PO` (resp. `MAL-1PO`) asks for a linear extension T_1 of P_1 that yields a maximum number of common intervals (resp. maximum number of adjacencies) between any linear extension of P_1 and T_2. Observe that there is no loss of generality in assuming here that $T_2 = \mathrm{Id}$. The third problem is concerned with confronting two partially ordered genomes: given two partially ordered genomes P_1 and P_2, problem `MAL-2PO` asks for a linear extension T_1 of P_1 and a linear extension T_2 of P_2 s.t. the number of adjacencies between T_1 and T_2 is maximized among all linear extensions of P_1 and P_2. Note that problems `MCIL-1PO`, `MAL-1PO` and `MAL-2PO` have been proved to be **NP**-hard in [4].

3 An exact $(0, 1)$-linear programming approach

We present in this section an exact generic approach. The main idea is to transform our problems into $(0, 1)$-linear programs [7] and use a powerful solver to ob-

tain optimal solutions. All computations conducted here use the `minisat+` solver [5]. In order to solve problems `MCIL-1PO`, `MAL-1PO` and `MAL-2PO`, we present the $(0,1)$-linear programs called `CI-1PO`, `Adj-1PO` and `Adj-2PO`, respectively. Programs `CI-1PO` and `Adj-1PO` take as input a partial order P_1 while `Adj-2PO` takes two partial orders P_1 and P_2. We first present the common part of these three programs and next give a complete description of each program. Then, we present some data reduction rules for reducing the size of the programs.

The common part of the three programs. We present here the common part of programs `CI-1PO`, `Adj-1PO` and `Adj-2PO`. Let P_1 and P_2 be two partially ordered genomes over $[0:n+1]$. Fix x to be 1 if we consider either `CI-1PO` or `Adj-1PO`, or 1 or 2 if we consider `Adj-2PO`. For each problem, we seek for a linear extension T_x of P_x that maximizes the number of common intervals or the number of adjacencies. Our programs are divided into two parts: (i) definition of a linear extension T_x, and (ii) maximization of the measure. The first part is common to the three programs whereas the second part is problem dependent and needs specific variables and constraints. To define a linear extension of a partially ordered genome, we use the same set $\mathcal{A}^x = \{a^x_{\mathtt{g},i} \ : \ \mathtt{g} \in [0:n+1] \text{ and } i \in [0:n+1]\}$ of boolean variables. For each $a^x_{\mathtt{g},i} \in \mathcal{A}^x$, $a^x_{\mathtt{g},i} = 1$ iff $T_x(\mathtt{g}) = i$, i.e., $\mathtt{g}$ is at position i in the resulting linear extension. To this aim, we define the three constraints presented in Figure 1:

- **(C.a)** ensures that each gene is assigned to exactly one position in T_x,
- **(C.b)** ensures that no two genes are assigned to the same position in T_x,
- **(C.c)** checks genes order in T_x. Let $\mathtt{g}_1$ and $\mathtt{g}_2$ be two genes of P_x such that $\mathtt{g}_1 \prec_x \mathtt{g}_2$. We must certainly check that $\mathtt{g}_1$ precedes $\mathtt{g}_2$ in T_x. Thus, for all $0 \leq j < i \leq n+1$ we impose that $a^x_{\mathtt{g}_1,i} = 0$ or $a^x_{\mathtt{g}_2,j} = 0$.

Common constraints of programs `CI-1PO`, `Adj-1PO` and `Adj-2PO`.

Constraints:

`C.a` $\forall x \in \{1,2\}, \forall\ 0 \leq \mathtt{g} \leq n+1, \sum_{0 \leq i \leq n+1} a^x_{\mathtt{g},i} = 1$

`C.b` $\forall x \in \{1,2\}, \forall\ 0 \leq i \leq n+1, \sum_{0 \leq \mathtt{g} \leq n+1} a^x_{\mathtt{g},i} = 1$

`C.c` $\forall x \in \{1,2\}, \forall\ 0 \leq \mathtt{g}_1 \leq n+1,\ 0 \leq \mathtt{g}_2 \leq n+1,\ \mathtt{g}_1 \prec_x \mathtt{g}_2,\ 0 < j \leq i \leq n+1,$
$x a^x_{\mathtt{g}_1,i} + a^x_{\mathtt{g}_2,j} \leq 1$

Fig. 1. Common constraints for `CI-1PO`, `Adj-1PO` and `Adj-2PO`. A gene $\mathtt{g}$ is at position i in T_x iff $a^x_{\mathtt{g},i}$. Constraints `C.a` to `C.c` ensure that T_x is a valid linear extension of P_x, $x \in \{1,2\}$.

Now, we present specific variables and constraints for each problem.

Confronting a partially ordered genome with a totally ordered genome: maximizing the number of common intervals. We give here a $(0,1)$-linear program called `CI-1PO` that computes a linear extension T_1 of a partially ordered genome P_1 that maximizes the number of common intervals between T_1 and Id. Recall that common intervals do not take into account the sign of the genes. Program `CI-1PO` is given in Figure 2.

We describe now the variables used in `CI-1PO`. Let $\mathtt{g} \in [0 : n+1]$ be a gene, $i \in [0 : n+1]$ be a position and $t \in [0 : n+1]$. After having defined the total order T_1 with the set $\mathcal{A}^1$, we define two sets of boolean variables: $\mathcal{B} = \{b_{\mathtt{g},i,t} \ : \ \mathtt{g} \in [0 : n+1],\ i \in [0 : n+1] \text{ and } t \in [0 : n+1]\}$ and $\mathcal{C} = \{c_{\mathtt{g},i,t} \ : \ \mathtt{g} \in [0 : n+1],\ i \in [0 : n+1] \text{ and } t \in [0 : n+1]\}$. For each $b_{\mathtt{g},i,t} \in \mathcal{B}$, we have $b_{\mathtt{g},i,t} = 1$ iff $T_1(\mathtt{g}) \in [i : i+t]$, i.e., $\mathtt{g}$ is located between positions i and $i+t$ in T_1. The set $\mathcal{C}$ corresponds to the set of all possible common intervals. For each $c_{\mathtt{g},i,t} \in \mathcal{C}$, we have $c_{\mathtt{g},i,t} = 1$ iff, for all $k \in [0 : t]$, we have $T_1(\mathtt{g}+k) \in [i : i+t]$, i.e., the set of genes $\mathtt{g}+k$ ($k \in [0 : t]$) is an interval of T_1. In this case, we certainly have a common interval between T_1 and the identity. Therefore, the objective function is the sum of all variables $c_{\mathtt{g},i,t} \in \mathcal{C}$.

Constraints **(C.01)** to **(C.05)** are needed to ensure correctness of `CI-1PO`. First, we obviously need constraints **(C.01)** to **(C.03)**, that correspond to the common constraints **(C.a)** to **(C.c)**, with $x = 1$. Then, we add constraint **(C.04)** to check variables $b_{\mathtt{g},i,t}$: $b_{\mathtt{g},i,t} = 1$ iff there exists j in $[0 : t]$ such that $a_{\mathtt{g},i+j} = 1$. Finally, we add constraint **(C.05)** to check variables $c_{\mathtt{g},i,t}$: $c_{\mathtt{g},i,t} = 1$ iff, for all k in $[0 : t]$, we have $b_{\mathtt{g}+k,i,t} = 1$.

Program `CI-1PO`

Objective: `Maximize` $\displaystyle\sum_{0\le \mathtt{g}\le n+1}\ \sum_{0\le i\le n+1}\ \sum_{0\le t\le n+1-i} c_{\mathtt{g},i,t}$

Constraints:

`C.01` $\forall\ 0 \le \mathtt{g} \le n+1,\ \displaystyle\sum_{0\le i\le n+1} a^1_{\mathtt{g},i} = 1$

`C.02` $\forall\ 0 \le i \le n+1,\ \displaystyle\sum_{0\le \mathtt{g}\le n+1} a^1_{\mathtt{g},i} = 1$

`C.03` $\forall\ 0 \le \mathtt{g}_1 \le n+1, 0 \le \mathtt{g}_2 \le n+1, \mathtt{g}_1 \prec_1 \mathtt{g}_2, 0 < j \le i \le n+1, a^1_{\mathtt{g}_1,i} + a^1_{\mathtt{g}_2,j} < 1$

`C.04` $\forall\ 0 \le \mathtt{g} \le n+1,\ 0 \le i \le n+1,\ 0 \le t \le n+1-i,\ \displaystyle\sum_{0\le j\le t} a^1_{\mathtt{g},i+j} - b_{\mathtt{g},i,t} = 0$

`C.05` $\forall\ 0 \le \mathtt{g} \le n+1,\ 0 \le i \le n+1,\ 0 \le t \le n+1-i,$

$$\sum_{0\le k\le t} b_{\mathtt{g}+k,i,t} - c_{\mathtt{g},i,t} < t+1 \text{ and } \sum_{0\le k\le t} b_{\mathtt{g}+k,i,t} - t.c_{\mathtt{g},i,t} \ge 0$$

Fig. 2. Program `CI-1PO` computes a linear extension L of a partial order maximizing the maximum number of common intervals between L and the identity.

Confronting a partially ordered genome with a totally ordered genome: maximizing the number of adjacencies. We now give a $(0,1)$-linear program (`Adj-1PO`) that computes a linear extension T_1 of a partial order P_1 that maximizes the number of adjacencies between T_1 and Id. Oppositely to common intervals, the sign of genes is now relevant. Program `Adj-1PO` is presented in Figure 3.

Let $\mathtt{g} \in [0 : n+1]$ be a gene, and $i \in [0 : n+1]$ be a position. After having defined the total order T_1 with the set of variables $\mathcal{A}^1$, we define the set $\mathcal{D} = \{d_{\mathtt{g},i} \ : \ \mathtt{g} \in [0 : n+1[\text{ and } i \in [0 : n+1]\}$ of boolean variables that corresponds to the set of possible adjacencies. For each $d_{\mathtt{g},i} \in \mathcal{D}$ ($\mathtt{g} \neq n+1$), we have $d_{\mathtt{g},i} = 1$ iff (i) $T_1(\mathtt{g}) = i$ and (ii) $\mathtt{g}$ and $(\mathtt{g}+1)$ create an adjacency. Therefore, we define the objective function as the sum of all variables $d_{\mathtt{g},i} \in \mathcal{D}$.

We now define the constraints **(C'.01)** to **(C'.06)** to ensure the validity of program `Adj-1PO`. We first add constraints **(C'.01)** to **(C'.03)**, that correspond to the common constraints **(C.a)** to **(C.c)** with $x = 1$, to obtain a valid linear extension T_1. Next, we add constraints **(C'.04)** to connect the assignment of variables in $\mathcal{D}$ to the assignment of variables in $\mathcal{A}^1$. To this aim, for each $d_{\mathtt{g},i} \in \mathcal{D}$, $\mathtt{g} \neq n+1,\ i \notin \{0, n+1\}$, we have $d_{\mathtt{g},i} = 1$ iff the two following conditions hold: (i) $a^1_{\mathtt{g},i} = 1$ (i.e., $\mathtt{g}$ is at the position i in T_1) and (ii) $a^1_{\mathtt{g}+1,i+1} = 1$ (case 1) or $a^1_{\mathtt{g}+1,i-1} = 1$ (case 2) (depending on the sign of $\mathtt{g}$ and $\mathtt{g}+1$). Finally, constraints **(C'.05)** and **(C'.06)** check variables $d_{\mathtt{g},0}$ and $d_{\mathtt{g},n+1}$ that correspond to the two possible adjacencies located at the extremities. These two constraints are defined as constraints **(C'.04)**, **(C'.05)** (case 1 only) and **(C'.06)** (case 2 only).

Confronting two partially ordered genomes: maximizing the number adjacencies. The $(0,1)$-linear program `Adj-2PO` proposed here computes two linear extensions T_1 and T_2 of two partially ordered genomes P_1 and P_2 such that the number of adjacencies between T_1 and T_2 is maximized. Recall that the sign of the genes is relevant here. Oppositely to `MAL-1PO` where one genome is totally ordered, we do not know the relation between two genes that will eventually create an adjacency. Program `Adj-2PO` is given in Figure 4.

Let $\mathtt{g}_1 \in [0 : n+1]$ and $\mathtt{g}_2 \in [0 : n+1]$ be two genes, and let $i \in [0 : n+1]$ and $j \in [0 : n+1]$ be two positions. After having defined the total orders T_1 and T_2 with the sets of variables $\mathcal{A}^1$ and $\mathcal{A}^2$, we introduce the set of variables $\mathcal{E} = \{e_{\mathtt{g}_1,i,j,\mathtt{g}_2} \ : \ \mathtt{g}_1 \in [0:n+1], i \in [0:n+1], j \in [0:n+1] \text{ and } \mathtt{g}_2 \in [0:n+1]\}$ that correspond to the possible adjacencies. A boolean variable $e_{\mathtt{g}_1,i,j,\mathtt{g}_2} \in \mathcal{E}$ has of four indices: genes $\mathtt{g}_1$ and $\mathtt{g}_2$, and positions i and j. For each $e_{\mathtt{g}_1,i,j,\mathtt{g}_2} \in \mathcal{E}$, we have $e_{\mathtt{g}_1,i,j,\mathtt{g}_2} = 1$ iff (i) genes $\mathtt{g}_1$ and $\mathtt{g}_2$ create an adjacency and (ii) $T_1(\mathtt{g}_1) = i$, $T_2(\mathtt{g}_1) = j$, and $T_1(\mathtt{g}_2) = i+1$. The objective function is defined as the sum of variables $e_{\mathtt{g}_1,i,j,\mathtt{g}_2} \in \mathcal{E}$.

We define constraints **(C".01)** to **(C".09)** to check correctness of program `Adj-2PO`. First, we use again the common constraints **(C.a)** to **(C.c)**. Here, we must use these constraints for x in $\{1, 2\}$ in order to define two linear extensions. We refer to these constraints as constraints **(C".01)** to **(C".06)**. Then, we check the assignment of variables $e_{\mathtt{g}_1,i,j,\mathtt{g}_2}$ for $i \neq n+1$ and $j \notin \{0, n+1\}$ with constraint **(C".07)**. To this aim, for each $e_{\mathtt{g}_1,i,j,\mathtt{g}_2}$, we have $e_{\mathtt{g}_1,i,j,\mathtt{g}_2} = 1$ iff the two following conditions hold: (i) $a^1_{\mathtt{g}_1,i} = 1$, $a^2_{\mathtt{g}_1,j} = 1$, and $a^1_{\mathtt{g}_2,i+1} = 1$, and (ii) $a^2_{\mathtt{g}_2,j+1} = 1$

Program `Adj-1PO`

Objective: `Maximize` $\sum_{0\leq \mathtt{g}\leq n} \sum_{0\leq i\leq n+1} d_{\mathtt{g},i}$

Constraints:

`C'.01` $\forall\ 0 \leq \mathtt{g} \leq n+1,\ \sum_{0\leq i\leq n+1} a^1_{\mathtt{g},i} = 1$

`C'.02` $\forall\ 0 \leq i \leq n+1,\ \sum_{0\leq \mathtt{g}\leq n+1} a^1_{\mathtt{g},i} = 1$

`C'.03` $\forall\ 0 \leq \mathtt{g}_1 \leq n+1,\ 0 \leq \mathtt{g}_2 \leq n+1,\ \mathtt{g}_1 \prec_0 \mathtt{g}_2,\ 0 < j \leq i \leq n+1,$

$$a^1_{\mathtt{g}_1,i} + a^1_{\mathtt{g}_2,j} \leq 1$$

`C'.04` $\forall\ 0 \leq \mathtt{g} \leq n, 1 \leq i \leq n,$

if $s_1(\mathtt{g}) =$ " $+$ " and $s_1(\mathtt{g}+1) =$ " $+$ " then $a^1_{\mathtt{g},i} + a^1_{\mathtt{g}+1,i+1} - d_{\mathtt{g},i} \leq 1$ and $3d_{\mathtt{g},i} - a^1_{\mathtt{g},i} - a^1_{\mathtt{g}+1,i+1} \leq 1$

if $s_1(\mathtt{g}) =$ " $-$ " and $s_1(\mathtt{g}+1) =$ " $-$ " then $a^1_{\mathtt{g},i} + a^1_{\mathtt{g}+1,i-1} - d_{\mathtt{g},i} \leq 1$ and $3d_{\mathtt{g},i} - a^1_{\mathtt{g},i} - a^1_{\mathtt{g}+1,i+1} \leq 1$

else $d_{\mathtt{g},i} = 0$

`C'.05` $\forall\ 0 \leq \mathtt{g} \leq n,$

if $s_1(\mathtt{g}) =$ " $+$ " and $s_1(\mathtt{g}+1) =$ " $+$ " then $a^1_{\mathtt{g},0} + a^1_{\mathtt{g}+1,1} - d_{\mathtt{g},0} \leq 1$ and $3d_{\mathtt{g},0} - a^1_{\mathtt{g},0} - a^1_{\mathtt{g}+1,1} \leq 1$

else $d_{\mathtt{g},0} = 0$

`C'.06` $\forall\ 0 \leq \mathtt{g} \leq n,$

if $s_1(\mathtt{g}) =$ " $-$ " and $s_1(\mathtt{g}+1) =$ " $-$ " then $a^1_{\mathtt{g},n+1} + a^1_{\mathtt{g}+1,n} - d_{\mathtt{g},n+1} \leq 1$ and $3d_{\mathtt{g},n+1} - a^1_{\mathtt{g},n+1} - a^1_{\mathtt{g}+1,n} \leq 1$

else $d_{\mathtt{g},n+1} = 0$

Fig. 3. Program `Adj-1PO` computes a linear extension of a partially ordered genome that maximizes the number of adjacencies between this linear extension and the identity.

(case 1) or $a^2_{\mathtt{g}_2,j-1} = 1$ (case 2) (these two cases depend on the sign of $\mathtt{g}_1$ and $\mathtt{g}_2$). Finally, we add constraints **(C".08)** and **(C".09)** to ensure the validity of variables $e_{\mathtt{g}_1,i,0,\mathtt{g}_2}$ and $e_{\mathtt{g}_1,i,n+1,\mathtt{g}_2}$ that correspond to possible adjacencies located at the extremities. These two constraints are similar to **(C".07)** but constraints **(C".08)** is for case 1 only and **(C".09)** is for case 2 only.

Speeding-up the program. Program `CI-1PO` has $O(n^3)$ variables and $O(n^3)$ constraints. Program `Adj-1PO` has $O(n^2)$ variables and $O(n^2)$ constraints, while program `Adj-2PO` has $O(n^4)$ variables and $O(n^4)$ constraints. In order to speed-up the running time of the programs, we present here some simple rules for reducing the number of variables and constraints involved in the programs or for computing in a fast preprocessing step some of the variables.

Program `Adj-2PO`

Objective: `Maximize` $\sum_{0\leq \mathtt{g}\leq n+1}\ \sum_{0\leq i\leq n+1}\ \sum_{0\leq j\leq n+1}\ \sum_{0\leq y\leq n+1} e_{\mathtt{g},i,j,y}$

Constraints:

`C''.01` $\forall\, 0\leq \mathtt{g}\leq n+1,\ \sum_{0\leq i\leq n+1} a^1_{\mathtt{g},i}=1$

`C''.02` $\forall\, 0\leq i\leq n+1,\ \sum_{0\leq \mathtt{g}\leq n+1} a^1_{\mathtt{g},i}=1$

`C''.03` $\forall\, 0\leq \mathtt{g}_1\leq n+1, 0\leq \mathtt{g}_2\leq n+1, \mathtt{g}_1\prec_0 \mathtt{g}_2, 0<j\leq i\leq n+1, a^1_{\mathtt{g}_1,i}+a^1_{\mathtt{g}_2,j}\leq 1$

`C''.04` $\forall\, 0\leq \mathtt{g}\leq n+1,\ \sum_{0\leq i\leq n+1} a^2_{\mathtt{g},i}=1$

`C''.05` $\forall\, 0\leq i\leq n+1,\ \sum_{0\leq \mathtt{g}\leq n+1} a^2_{\mathtt{g},i}=1$

`C''.06` $\forall\, 0\leq \mathtt{g}_1\leq n+1, 0\leq \mathtt{g}_2\leq n+1, \mathtt{g}_1\prec_1 \mathtt{g}_2, 0<j\leq i\leq n+1, a^2_{\mathtt{g}_1,i}+a^2_{\mathtt{g}_2,j}\leq 1$

`C''.07` $\forall\, 0\leq \mathtt{g}_1\leq n+1, 0\leq i\leq n+1, 1\leq j\leq n, 0\leq \mathtt{g}_2\leq n+1,$
if $s_1(\mathtt{g}_1)=s_2(\mathtt{g}_1)$ and $s_1(\mathtt{g}_2)=s_2(\mathtt{g}_2)$ then
$a^1_{\mathtt{g}_1,i}+a^2_{\mathtt{g}_1,j}+a^1_{\mathtt{g}_2,i+1}+a^2_{\mathtt{g}_2,j+1}-4e_{\mathtt{g}_1,i,j,\mathtt{g}_2}\geq 0$
and $a^1_{\mathtt{g}_1,i}+a^2_{\mathtt{g}_1,j}+a^1_{\mathtt{g}_2,i+1}+a^2_{\mathtt{g}_2,j+1}-4e_{\mathtt{g}_1,i,j,\mathtt{g}_2}\leq 3$
if $s_1(\mathtt{g}_1)=-s_2(\mathtt{g}_1)$ and $s_1(\mathtt{g}_2)=-s_2(\mathtt{g}_2)$ then
$a^1_{\mathtt{g}_1,i}+a^2_{\mathtt{g}_1,j}+a^1_{\mathtt{g}_2,i+1}+a^2_{\mathtt{g}_2,j-1}-4e_{\mathtt{g}_1,i,j,\mathtt{g}_2}\geq 0$
and $a^1_{\mathtt{g}_1,i}+a^2_{\mathtt{g}_1,j}+a^1_{\mathtt{g}_2,i+1}+a^2_{\mathtt{g}_2,j-1}-4e_{\mathtt{g}_1,i,j,\mathtt{g}_2}\leq 3$
else $e_{\mathtt{g}_1,i,j,\mathtt{g}_2}=0$

`C''.08` $\forall\, 0\leq \mathtt{g}_1\leq n+1, 0\leq i\leq n+1, 0\leq \mathtt{g}_2\leq n+1,$
if $s_1(\mathtt{g}_1)=s_2(\mathtt{g}_1)$ and $s_1(\mathtt{g}_2)=s_2(\mathtt{g}_2)$ then
$a^1_{\mathtt{g}_1,i}+a^1_{\mathtt{g}_1,0}+a^2_{\mathtt{g}_2,i+1}+a^2_{\mathtt{g}_2,1}-4e_{\mathtt{g}_1,i,0,\mathtt{g}_2}\geq 0$
and $a^1_{\mathtt{g}_1,i}+a^1_{\mathtt{g}_1,0}+a^2_{\mathtt{g}_2,i+1}+a^2_{\mathtt{g}_2,1}-4e_{\mathtt{g}_1,i,0,\mathtt{g}_2}\leq 3$
else $e_{\mathtt{g}_1,i,0,\mathtt{g}_2}=0$

`C''.09` $\forall\, 0\leq \mathtt{g}_1\leq n+1, 0\leq i\leq n+1, 0\leq \mathtt{g}_2\leq n+1,$
if $s_1(\mathtt{g}_1)=-s_2(\mathtt{g}_1)$ and $s_1(\mathtt{g}_2)=-s_2(\mathtt{g}_2)$ then
$a^1_{\mathtt{g}_1,i}+a^1_{\mathtt{g}_1,n+1}+a^2_{\mathtt{g}_2,i+1}+a^2_{\mathtt{g}_2,n}-4e_{\mathtt{g}_1,i,n+1,\mathtt{g}_2}\geq 0$
and $a^1_{\mathtt{g}_1,i}+a^1_{\mathtt{g}_1,n+1}+a^2_{\mathtt{g}_2,i+1}+a^2_{\mathtt{g}_2,n}-4e_{\mathtt{g}_1,i,n+1,\mathtt{g}_2}\leq 3$
else $e_{\mathtt{g}_1,i,n+1,\mathtt{g}_2}=0$

Fig. 4. Program `Adj-2PO` computes two linear extensions of two partial orders maximizing the number of adjacencies.

Restricted position. In T_x, for each gene $\mathtt{g}$, $T_x(\mathtt{g})$ must be in $POS_x(\mathtt{g})$. According to this remark, some variables can be pre-computed:

- $\forall\, x\in\{1,2\}$, $\forall\, a^x_{\mathtt{g},i}\in\mathcal{A}^x$, $a^x_{\mathtt{g},i}=0$ if $i\notin POS_x(\mathtt{g})$;
- $\forall\, b_{\mathtt{g},i,t}\in\mathcal{B}$, $b_{\mathtt{g},i,t}=0$ if $T_1(\mathtt{g})$ cannot be in $[i:i+t]$, and $b_{\mathtt{g},i,t}=1$ if $T_1(\mathtt{g})$ is necessarily in $[i:i+t]$;

- $\forall\, d_{\mathtt{g},i} \in \mathcal{D}$, $d_{\mathtt{g},i} = 0$ if $i \notin POS_1(\mathtt{g})$;
- $\forall\, e_{\mathtt{g}_1,i,j,\mathtt{g}_2} \in \mathcal{E}$, $e_{\mathtt{g}_1,i,j,\mathtt{g}_2} = 0$ if one of the four following conditions holds: (i) $i \notin POS_1(\mathtt{g}_1)$, (ii) $i+1 \notin POS_1(\mathtt{g}_2)$, (ii) $i+1 \notin POS_2(\mathtt{g}_2)$ and $s_1(\mathtt{g}_1) = s_1(\mathtt{g}_2)$, (iv) $i-1 \notin POS_2(\mathtt{g}_2)$ and $s_1(\mathtt{g}_1) = -s_1(\mathtt{g}_2)$.

For each gene $\mathtt{g}$ that is i-$nail_x$ (i.e., $\mathtt{g}$ is necessarily at a known position i in the linear extension), some variables can also be pre-computed:

- $\forall\, x \in \{1,2\}$, $\forall\, a^x_{\mathtt{g},i} \in \mathcal{A}^x$, $a^x_{\mathtt{g},i} = 1$ if $\mathtt{g}$ is i-$nail_x$;
- $\forall\, d_{\mathtt{g},i} \in \mathcal{D}$, $d_{\mathtt{g},i} = 1$ if gene $\mathtt{g}$ is i-$nail_1$ and if one of the following conditions is satisfied: (i) gene $(\mathtt{g}+1)$ is $(i+1)$-$nail_2$ with $s_1(\mathtt{g}) =$ "+" and $s_1(\mathtt{g}+1) =$ "+", or (ii) gene $(\mathtt{g}+1)$ is $(i-1)$-$nail_2$ with $s_1(\mathtt{g}) =$ " $-$ " and $s_1(\mathtt{g}+1) =$ " $-$ ".
- $\forall\, e_{\mathtt{g}_1,i,j,\mathtt{g}_2} \in \mathcal{E}$, $e_{\mathtt{g}_1,i,j,\mathtt{g}_2} = 1$ if gene $\mathtt{g}_1$ is i-$nail_1$ and j-$nail_2$, gene $\mathtt{g}_2$ is a $(i+1)$-$nail_1$, and one of the two following conditions holds: (i) $\mathtt{g}_2$ is $(j+1)$-$nail_2$ with $s_1(\mathtt{g}_1) = s_1(\mathtt{g}_2)$, (ii) $\mathtt{g}_2$ is $(j-1)$-$nail_2$ with $s_1(\mathtt{g}_1) = -s_1(\mathtt{g}_2)$.

Specific rules for `CI-1PO`. By definition, each interval of size 1 and $n+2$ is a common interval. Therefore, $c_{\mathtt{g},i,0} = 1$ and $c_{\mathtt{g},0,n+2} = 1$ for $0 \leq \mathtt{g} \leq n+1$, and we add $n+2$ to the objective function. Also, we can delete all variables $c_{\mathtt{g},i,0}$ and $c_{\mathtt{g},i,n+1}$ in the program as well as their related constraints. The second reduction consists in removing some variables $c_{\mathtt{g},i,t}$. We do not generate a variable $c_{\mathtt{g},i,t}$ and its related constraints if $c_{\mathtt{g},i,t} = 1$ is not possible. Indeed, if there exists an integer $k \in [0:t]$ such that $T_1(\mathtt{g}+k)$ cannot be in $[i:i+t]$ (i.e., gene $\mathtt{g}+k$ cannot be between i and $i+t$ in the linear extension), then the interval $\{\mathtt{g}, \mathtt{g}+1, \ldots, \mathtt{g}+t\}$ is certainly not a common interval.

Specific rules for `Adj-1PO`. Constraint `C'.06` defines the adjacency created by genes $\mathtt{g}$ and $(\mathtt{g}+1)$ at positions $n+1$ and n, respectively, in T_1. Since we have artificially added gene $n+1$ at the end of partial order, then the gene at position $n+1$ is $n+1$. Therefore, $d_{\mathtt{g},n+1} = 0$ for $\mathtt{g}$ in $[0:n]$ and the constraints `C'.06` are deleted. Moreover, if no adjacency is possible between two genes that are *nails*, then the total order between these genes has no impact on the number of adjacencies. Specifically, let $\mathtt{g}_1$ and $\mathtt{g}_2$ be two genes such that $\mathtt{g}_1$ is i_1-$nail_1$ and $\mathtt{g}_2$ is i_2-$nail_1$ ($i_1 < i_2$). Notice that no adjacency can be create between the positions i_1 and i_2 between a linear extension of P_1 and the identity if, for each gene $\mathtt{g}_3$ such that $POS_1(\mathtt{g}_3) \subseteq [i_1:i_2]$, one of the three following conditions hold: (i) $s_1(\mathtt{g}_3) \neq s_1(\mathtt{g}_3+1)$, (ii) $s_1(\mathtt{g}_3) =$ " $+$ " and $s_1(\mathtt{g}_3+1) =$ " $+$ " and $(\mathtt{g}_3, \mathtt{g}_3+1)$ is not adjoining in P_1, or (iii) $s_1(\mathtt{g}_3) =$ " $-$ " and $s_1(\mathtt{g}_3+1) =$ " $-$ " and $(\mathtt{g}_3+1, \mathtt{g}_3)$ is not adjoining in P_1. Hence, if each such gene $\mathtt{g}_3$ satisfies at least one condition, we can define arbitrarily a total order between i_1 and i_2 without adding an adjacency.

4 Experimental results

Following the example of [4], we have tested our algorithms on simulated data to assess the performance of our programs for different parameters inherent to

partial orders. The rationale for this choice is two-fold. For one, the presented algorithms are quite complicated and using parameterized data allows us to tune the programs to best solve the problems. For another, using the dataset of [4] makes comparisons easier with previous works. Our linear program solver engine is powered by the `minisat+` solver [5]. All computations were carried out on a *Duo Intel* CPU GHz with 2 Gb of memory running under Linux. The reference data set is from [4]. Blin *et al.* have generated partial orders P_x according to three parameters: the *size* n, the *order rate* p that determines the number of adjoining in the expression, and the *gene distribution rule* q that corresponds to the probability of possible adjacencies with respect to the identity. We use 19 different instances for each triplet of parameters (n, p, q) with $n \in \{30, 40, 50, 60, 70, 80, 90\}$, $p \in \{0.7, 0.9\}$ and $q \in \{0.4, 0.6, 0.8\}$, so that we have 798 genomes for each program. The results of the three programs on this dataset are presented below. We evaluated two criteria: the running time and the given measure (number of common intervals or number of adjacencies) induced by the returned linear extension. The width of partial orders will be also considered. Then, we study the advantages of program `Adj-1PO` to `Adj-2PO` for the comparison of a partial order and the identity.

Results for CI-1PO. For program `CI-1PO`, the `minisat+` engine resolves 264 instances out of 342 inputs (77.2%) with $n \in \{30, 40, 50\}$, $p \in \{0.7, 0.9\}$, and $q \in \{0.4, 0.6, 0.8\}$ (Table 1) in less than 5 hours. The 83 remaining cases have been stopped after 30 minutes. We note that `CI-1PO` reaches its limits even for small instances. As shown in Table 2, the width of an instance is an important parameter that certainly contributes to the complexity of the instances. Indeed, we remark that the number of obtained results decreases according to the width of the instances. In the same way, the average running time increases with the width. We also notice that the width is correlated with the complexity of the problem, which is not surprising since combinatorial difficulty is clearly contained in the greatest sets of non-comparable genes.

Size	Number of results	Number of instances	Running time
30	95	114	$1h\ 48m$
40	85	114	$1h\ 19m$
50	79	114	$1h\ 44m$
Total	259	342	$4h\ 41m$

Table 1. Results of `CI-1PO` obtained in less than thirty minutes.

Results for Adj-1PO. We applied `Adj-1PO` on 19 genomes for each triplet (n, p, q) with $n \in \{30, 40, 50, 60, 70, 80, 90\}$, $p \in \{0.7, 0.9\}$ and $q \in \{0.4, 0.6, 0.8\}$. We obtained 778 results out of 798 $(97, 5\%)$ after 2 months and 13 days. Due to huge memory requirements, a Quadri Intel(R) Xeon(TM) CPU 3,00 GHz with

Width	1	2	3	4	5	6	7	8	9	10	11
Number of instances	5	63	63	50	44	31	26	18	13	7	9
Number of solved instances	5	63	63	44	36	26	15	5	1	1	0
% of solved instances	100%	100%	100%	88%	82%	84%	58%	28%	8%	15%	0%
Average running time (sec)	2	13	28	60	88	154	201	311	73	490	

Table 2. Impact of width for `CI-1PO`.

16Gb of memory running Linux was required for 14 of the runs. We note that parameter p has the largest impact on the running time. It is indeed significantly higher when $p = 0.7$ rather than when $p = 0.9$, i.e., when the partial orders have less adjoining. The width also affects the running time. Parameter q seems, however, to have no impact on the running time. For the 70 cases where the running time exceeds 1 hour, we note that the corresponding genomes contain 50 genes or more, with an order rate p equals to 0.7 (58 times out of 70), but without specific gene distribution (21 times 0.4, 27 times 0.6 and 22 times 0.8). Their width ranges from 5 to 22 and is on average equal to 11.6. For 19 genomes, the linear program could not been solved by `minisat+` because the number of constraints and variables is too large for this solver. For these genomes, we observe that their size is greater than or equal to 70, that p is equal to 0.7 and that q is variable (9 times 0.4, 3 times 0.6 and 8 times 0.8). Their width is also large: 17.5 on average (from 11 to 29), compared to 6 for all genomes.

Results for Adj-2PO. For each triplet, we compare 19 genomes by pairs. At the present time, we have obtained the results for the following triplets: $\{30, 0.7, 0.8\}$, $\{30, 0.9, 0.4\}$, $\{30, 0.9, 0.6\}$, $\{30, 0.9, 0.8\}$, $\{40, 0.9, 0.4\}$, $\{40, 0.9, 0.6\}$, $\{40, 0.9, 0.8\}$, $\{50, 0.9, 0.6\}$, $\{50, 0.9, 0.8\}$, $\{60, 0.9, 0.6\}$, $\{60, 0.9, 0.8\}$, and $\{70, 0.9, 0.8\}$. Before discussing the 11 triplets for which $p = 0.9$, let us look at the results obtained with the triplet $\{30, 0.7, 0.8\}$. The running time is clearly more important when $p = 0.7$ rather than $p = 0.9$ (2 days and 21 hours for the triplet $\{30, 0.7, 0.8\}$ and 40 minutes for the triplet $\{30, 0.9, 0.8\}$). Moreover, fewer results were obtained (90 results for the triplet $\{30, 0.7, 0.8\}$ and 171 for the triplet $\{30, 0.9, 0.8\}$). For the 11 triplets for which $p = 0.9$, we made 1881 comparisons (19 genomes compared with 18 genomes of the same triplet). We manage to obtain in 1668 results (i.e., 88.6%) in a little over 38 days. For the 213 unfinished runs, two types of problems have emerged: either the linear program has too many variables and constraints for the solver `minisat+`, or the memory requested is too large for the Duo Intel 2Gb of memory. The width clearly influences the running time. Indeed, when the sum of width of both partials orders P_1 and P_2 is less than 5, the average running time is 14 seconds, while it is of 56 minutes when this sum is more than 5. Parameter q has an impact on the number of adjacencies only.

Comparison between `Adj-1PO` and `Adj-2PO`. To evaluate `Adj-1PO`, we compare its running time with `Adj-2PO` when P_2 is the identity. For the 19 instances of each triplet (n, p, q) such that n is in $\{30, 40\}$, p is in $\{0.7, 0.9\}$ and q is in

$\{0.4, 0.6, 0.8\}$, the sum of running times is 2 hours and 30 minutes (minimum = 0.1 second, maximum = 15 minutes 25s) for `Adj-1PO` and 67 hours (minimum = 0.1 second, maximum = 14 hours 42m) for `Adj-2PO`. Clearly, `Adj-1PO` is much faster than `Adj-2PO` from the viewpoint of running time (a diminution of 96.3%). If we compare the running time for each case, we note that both programs have a running time more important for the same genome. We can infer that both programs are facing the same difficulties. This is not surprising in view of the similarities of variables and constraints of linear programming of `Adj-1PO` and `Adj-2PO`.

5 Conclusion and future works

Our results are quite preliminary and there is still a great amount of work to be done. Among other things, one can cite: (i) for each case, determine other strong and relevant rules for speeding-up the process by avoiding the generation of too many variables and constraints, (ii) generalize the program `CI-1PO` to compute the number of common intervals between two partial orders, and (iii) define and evaluate heuristics for these problems. Indeed, we do think that our approach is useful for providing exact reference results to which new developed heuristics can be compared.

References

1. S. Angibaud, G. Fertin, I. Rusu, A. Thévenin, and S. Vialette. Efficient tools for computing the number of breakpoints and the number of adjacencies between two genomes with duplicate genes. *J. Computational Biology*, 15(8):1093–1115, 2008.
2. S. Angibaud, G. Fertin, I. Rusu, and S. Vialette. A pseudo-boolean general framework for computing rearrangement distances between genomes with duplicates. *J. Computational Biology*, 14(4):379–393, 2007.
3. A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. In *Proc. 9th International Computing and Combinatorics Conference (COCOON '03)*, volume 2697 of *LNCS*, pages 68–79, 2003.
4. G. Blin, E. Blais, D. Hermelin, P. Guillon, M. Blanchette, and N. El-Mabrouk. Gene maps linearization using genomic rearrangement distances. *J. Computational Biology*, 14(4):394–407, 2007.
5. N. Eén and N. Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.
6. Z. Fu and T. Jiang. Computing the breakpoint distance between partially ordered genomes. *J. Bioinformatics and Computational Biology*, 5(5):1087–1101, 2007.
7. A. Schrijver. *Theory of Linear and Integer Programming*. J. Wiley & Sons, 1998.
8. T. Uno and M. Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26(2):290–309, 2000.
9. G. A. Watterson, W. J. Ewens, T. E. Hall, and A. Morgan. The chromosome inversion problem. *Journal of Theoretical Biology*, 99(1):1–7, 1982.
10. C. Zheng, A. Lenert, and D. Sankoff. Reversal distance for partially ordered genomes. *Bioinformatics*, 21(1):502–508, 2005.