Sorting Permutations by Reversals through Branch-and-Price

Alberto Caprara[*], Giuseppe Lancia[+] and See-Kiong Ng[o]

[*] DEIS, University of Bologna, Italy

[+] DEI, University of Padova, Italy

[o] Smithkline Beecham Pharmaceuticals R&D, Bioinformatics, UK

July 1999

Abstract

We describe an exact algorithm for the problem of sorting a permutation by the minimum number of reversals, originating from evolutionary studies in molecular biology. Our approach is based on an integer linear programming formulation of a graph-theoretic relaxation of the problem, calling for a decomposition of the edge set of a bicolored graph into the maximum number of alternating cycles. The formulation has one variable for each alternating cycle, and the associated linear programming relaxation is solved by column generation.

A major advantage with respect to previous approaches is that the subproblem to face in the column generation phase requires no longer the solution of min-cost general matching problems, but of min-cost *bipartite* matching problems. Experiments show that there is a tremendous speed-up in going from general matching to bipartite matching, although the best-known algorithms for the two problems have the same theoretical worst-case complexity. We also show the worst-case ratio between the lower bound value obtained by our new method and previous ones.

We illustrate the effectiveness of our approach through extensive computational experiments. In particular, we can solve to proven optimality the largest real-world instances from the literature in few seconds, and the other (smaller) real-world instances within few milliseconds on a workstation. Moreover, we can solve to optimality random instances with $n = 100$ within 3 seconds, and with $n = 200$ within 15 minutes, where n is the size of the permutation, whereas the size of the instances solvable by previous approaches was at most 100. We also describe a polynomial-time heuristic algorithm which consistently finds solutions within 2% of the optimum for random instances with n up to 1000.

Key words: sorting by reversals, branch-and-price, alternating-cycle decomposition, column generation, matching.

AMS subject classifications: 68Q25, 68R10, 05C45.

1 Introduction

Many among the most effective approaches to solve Combinatorial Optimization problems are based on the definition of an *Integer Linear Programming* (ILP) formulation and the solution of the associated *Linear Programming* (LP) relaxation. A key issue for the success of this scheme is the "strength" of the ILP formulation, i.e. the fact that the optimal solution values of the ILP and its LP relaxation are "reasonably close" to each other. It is often a challenging task to derive strong ILP formulations. In some relevant cases, the only way to come up with such formulations is to have a very large (typically exponential in the size of the problem instance) number of constraints and/or variables in the model: The Branch-and-Cut [28] and *Branch-and-Price* (B&P) [2] methodologies were specifically developed to handle these cases. Still, for many important problems in Combinatorial Optimization, no strong ILP formulation is known, and the most effective methods to tackle these problems rely on lower bounding procedures and heuristic algorithms based on combinatorial considerations. It is common belief that the derivation of a strong ILP formulation (if any exists) for these problems would lead to a significant increase of the size of the instances solvable to proven optimality within reasonable computing time. A notable example of this situation are most of the scheduling problems, in particular all the problems in the Shop family [26].

In this paper we describe a successful ILP-based approach to the solution of a problem which apparently does not have any strong ILP formulation. A main idea of the approach is not to insist in formulating the original problem itself as an ILP, but to work with a (natural) ILP formulation of a combinatorial relaxation. We show how a B&P algorithm for this relaxation can easily be adapted to yield an effective exact algorithm for the original problem.

B&P approaches deal with the case in which the strong ILP formulations used have exponentially many variables, handling their LP relaxations by column generation techniques. A typical drawback of this approach is that an upper bound on the integer optimum is available only at the end of the procedure, namely when none of the variables has positive reduced cost, supposing the ILPs (and LPs) to be solved are in maximization form. (In fact, the process can be stopped slightly earlier [13], but this does not change the situation substantially.) Hence, in order to get an upper bound, the column generation problem is typically solved "exactly" a few times during the solution of the LP since, if no heuristic method is able to find a variable with positive reduced cost, one has to guarantee in some way that no such variable exists. This is clearly a drawback if the column generation problem is NP-complete (see e.g. [34], [27]), as one has to resort to enumerative methods, and may turn out to be inconvenient also when the latter problem can be solved in polynomial time.

In our case, the column generation problem for the LP relaxation of the ILP formulation mentioned above requires solving one or more (nonbipartite) min-cost perfect matching problems, as shown in [8], and the solution of these matching problems turns out to be the computational bottleneck of the overall method. To overcome this drawback, we propose a novel "heuristic" approach to the coulmn generation problem, which considers an enlarged set of variables. The definition of this enlarged set is inspired by the availability of an effective procedure to generate variables in this set. Namely, instead of solving some (nonbipartite) matching problems, we can solve some *bipartite* min-cost perfect matching problems, which requires a much smaller computing time in practice. The whole B&P approach works because the "new" variables in the formulation can take only fractional values when they are positive, and hence are forced to be 0 in any integer solution.

We deal with the problem of *Sorting a permutation By Reversals* (SBR), which is defined as follows. Let $\pi = (\pi_1 \ \dots \ \pi_n)$ be a permutation of $\{1, \dots, n\}$, and denote by ι the identity permutation $(1\ 2\ \dots\ n-1\ n)$. A *reversal* of the interval (i, j) is an inversion of the subsequence $\pi_i \ \dots \ \pi_j$ of π, represented by the permutation $\rho = (1 \ \dots \ i-1 \ \ j \ j-1 \ \dots \ i+1 \ i \ \ j+1 \ \dots \ n)$. Composition of π with ρ yields $\pi\rho = (\pi_1 \ \dots \ \pi_{i-1} \ \pi_j \ \pi_{j-1} \ \dots \ \pi_{i+1} \ \pi_i \ \pi_{j+1} \ \dots \ \pi_n)$ where elements $\pi_i, \dots, \pi_j$ have been reversed. Given a permutation π, SBR calls for a shortest sequence of reversals $\rho_1, \ldots, \rho_{d(\pi)}$ such that $\pi\rho_1 \ldots \rho_{d(\pi)} = \iota$. The optimal solution value $d(\pi)$ is called the *reversal distance* of π.

SBR originates from the problem of computing the evolutionary distance between two different genomes in computational molecular biology. In this case, a permutation may represent the linear order of a set of genes along a specific chromosome or of a set of markers along mitochondrial DNA. Looking at whole genomes has many advantages over comparing actual nucleotide sequences. In fact, it is often found that the order and number of genes is preserved far more easily than their content, i.e. the DNA sequence (see [14, 35]). For this reason, modern evolutionary studies in molecular biology are based on analyzing large-scale rearrangements that may turn a genome into another, such as inversions, transpositions, deletions, insertions and duplications of DNA fragments. Such genomic rearrangements are relatively rare events. Furthermore, inversions are the most frequent among them. Therefore SBR, which amounts to determine the minimum number of inversions that must have occurred in the evolution from a genome into another, is particularly relevant in evolutionary studies (see [31, 30]).

Let the order of the genes in two homologous chromosomes of different organisms with n genes be represented by two permutations π and τ of $\{1, \ldots, n\}$. An inversion of the segment comprising the genes from the i-th to the j-th corresponds to a reversal of the interval (i, j). A shortest sequence of reversals needed to transform π into τ is then clearly equal to an optimal solution of SBR on $\tau^{-1}\pi$.

With the more general aim of reconstructing an evolutionary tree (see e.g. [32]), SBR may be the subproblem to be solved to evaluate the distance between two species in the tree. In this respect, it would be convenient to have very fast (in practice) algorithms to solve it.

SBR has been widely studied in the last years, among others, by Kececioglu and Sankoff [25, 24], Bafna and Pevzner [1], Hannenhalli and Pevzner [19, 20], Caprara, Lancia and Ng [8], Berman and Hannenhalli [3], Irving and Christie [21], Tran [33], Kaplan, Shamir and Tarjan [22], Caprara [6, 7], Christie [10]. Most of these papers deal with the complexity and theoretical approximability of the problem. In particular, SBR was shown to be NP-hard in [6] and APX-hard in [4] (a problem is APX-hard if it does not have a polynomial time approximation scheme unless P = NP). References [25, 1, 10] present, respectively, polynomial-time 2-approximation, $\frac{7}{4}$-approximation, and $\frac{3}{2}$-approximation algorithms.

SBR was one of the main problems for the DIMACS Implementation Challenge in 1995 [12]. The exact solution of real-world and randomly-generated instances of the problem is reported in [25, 20, 8]. Here we present an exact B&P algorithm that widely outperforms these previous approaches and is capable of solving the largest real-world instances from molecular biology.

The paper is organized as follows. In Section 2 we survey known results in the literature, describing the combinatorial relaxation of SBR and the associated (natural) ILP formulation. Section 3 shows how to get a B&P algorithm for SBR that works with the LP relaxation of this formulation, whereas Section 4 explains our column generation scheme on an enlarged set of variables. Section 5 illustrates a few implementation issues that we introduced in order to speed-up the solution of the LP above, as well as to find quickly a feasible solution very close to the optimum. Finally, in Section 6 we present computational results both on real-world instances and on other instances from the literature, that show the effectiveness of our approach. In particular, we can solve to proven optimality the largest real-world instances

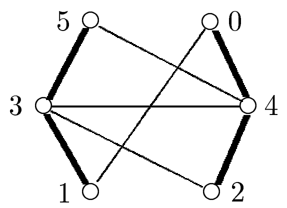

Figure 1: The breakpoint graph $G(\pi)$ associated with $\pi = (4\ 2\ 1\ 3)$. Gray edges are drawn as thin lines, black edges as thick lines.

from the literature in few seconds, and the other (smaller) real-world instances within few milliseconds on a workstation. Moreover, we can solve to optimality random instances with $n = 100$ within 3 seconds, and with $n = 200$ within 15 minutes, where n is the size of the permutation, whereas the size of the instances solvable by previous approaches was at most 100. We also describe a polynomial-time heuristic algorithm which consistently finds solutions within 2% of the optimum for random instances with n up to 1000.

2 SBR and Alternating-Cycle Decompositions

After unsuccessful attempts to find a reasonable ILP formulation for a Combinatorial Optimization problem, it is natural to start looking for combinatorial relaxations in order to find bounds on the optimal solution value. A very elegant graph-theoretic relaxation for SBR was introduced by Bafna and Pevzner [1] and Kececioglu and Sankoff [25].

Consider a permutation $\pi = (\pi_1\ \dots\ \pi_n)$ of $\{1, \dots n\}$. Throughout the paper, n will denote the number of elements of the permutation π considered. Following the description in [1], define the *breakpoint graph* $G(\pi) = (V, B \cup Y)$ of π as follows. Add to π the elements $\pi_0 := 0$ and $\pi_{n+1} := n + 1$, re-defining $\pi := (0\ \pi_1\ \dots\ \pi_n\ n + 1)$. Also, let the *inverse permutation* π^{-1} of π be defined by $\pi^{-1}_{\pi_i} := i$ for $i = 0, \dots, n + 1$. Let $V := \{0, \dots, n + 1\}$, where each node $v \in V$ represents an element of π. Graph $G(\pi)$ is *bicolored*, i.e. its edge set is partitioned into two subsets, each represented by a different color. B is the set of *black* edges, each of the form (π_i, π_{i+1}), for all $i \in \{0, \dots, n\}$ such that $|\pi_i - \pi_{i+1}| \neq 1$, i.e. elements which are in consecutive positions in π but not in the identity permutation ι. Such a pair π_i, π_{i+1} is called a *breakpoint* of π, and an element π_i is called a *singleton* if both π_{i-1}, π_i and π_i, π_{i+1} are breakpoints of π. Let $b(\pi) := |B|$ be the number of breakpoints of π. Y is the set of *gray* edges, each of the form $(i, i+1)$, for all $i \in \{0, \dots, n\}$ such that $|\pi^{-1}_i - \pi^{-1}_{i+1}| \neq 1$, i.e. elements which are in consecutive positions in ι but not in π. Note that each node $i \in V$ has either degree 0, 2 or 4, and has the same number of incident gray and black edges. Therefore, $|B| = |Y| (= b(\pi))$. Note also that there is no cycle formed by black (resp. grey) edges only. Figure 1 depicts the breakpoint graph associated with the permutation (4 2 1 3).

An *alternating cycle* of $G(\pi)$ is a cycle whose edges are alternately grey and black, without edge repetitions but possibly with node repetitions. Formally, an alternating cycle is a sequence of edges $b_1, y_1, b_2, y_2, \dots, b_m, y_m$, such that:

(i) $b_i \in B$, $y_i \in Y$ for $i = 1, \dots, m$;

(ii) b_i and y_j have a common endpoint for $i = j = 1, \dots, m$ and for $i = j + 1$, $j = 1, \dots, m$;

(iii) b_i and b_{i+1} (resp. y_i and y_{i+1}) have no common endpoint for $i = 1, \ldots, m$;

(iv) $b_i \neq b_j$ (resp. $y_i \neq y_j$) for $1 \leq i < j \leq m$;

where indices are understood to be modulo m. For example, edges $(0,4),(4,3),(3,1),(1,0)$ and $(4,2),(2,3),(3,5),(5,4)$ form alternating cycles in the graph of Figure 1.

An *alternating-cycle decomposition* of $G(\pi)$ is a collection of edge-disjoint alternating cycles, such that every edge of G is contained in exactly one cycle of the collection. It is easy to see that $G(\pi)$ always admits an alternating-cycle decomposition (recalling that an alternating cycle is allowed to visit the same node twice). In the graph of Figure 1, alternating cycles $(0,4),(4,3),(3,1),(1,0)$ and $(4,2),(2,3),(3,5),(5,4)$ form an alternating-cycle decomposition. Let $c(\pi)$ be the maximum cardinality of an alternating-cycle decomposition of $G(\pi)$. Bafna and Pevzner [1] (see also Kececioglu and Sankoff [25]) proved the following property

Theorem 1 *([1], [25]) For every permutation π, $d(\pi) \geq b(\pi) - c(\pi)$.*

Therefore $b(\pi) - c(\pi)$ gives a valid lower bound on the optimal solution value of SBR. The above discussion motivates the study of the *Alternating-Cycle Decomposition* (ACD) problem, which calls for a maximum-cardinality alternating-cycle decomposition of $G(\pi)$.

The intuition behind Theorem 1 is the following. Note that the identity permutation ι is the only one for which $b(\iota) = c(\iota) = 0$. First of all, it is easy to see that a reversal removes at most two breakpoints, i.e. any permutation ω obtained by applying one reversal to π satisfies $b(\omega) \geq b(\pi) - 2$. This implies $d(\pi) \geq \frac{b(\pi)}{2}$. In fact, whenever a reversal removes two breakpoints, the effect is also to "remove" a cycle of length 4 in the breakpoint graph, i.e. the permutation ω obtained by applying such a reversal to π satisfies $b(\omega) = b(\pi) - 2$ and $c(\omega) = c(\pi) - 1$. For instance, consider the reversal of the interval $(1,3)$ for $\pi = (4\ 2\ 1\ 3)$ represented in Figure 1. More generally, the proof of Theorem 1 shows that each reversal cannot decrease $b(\pi) - c(\pi)$ by more than 1; namely, in the "best case", given a maximum-cardinality alternating-cycle decomposition of $G(\pi)$, each reversal either removes two breakpoints and one cycle of length 4 in the decomposition, or it removes one breakpoint and makes one cycle of length ≥ 6 shorter.

A key question raised by the discussion above concerns the strength of lower bound $b(\pi) - c(\pi)$ on the optimal solution value $d(\pi)$. Since the very first computational experiments on SBR it was observed that this bound is very often equal to the optimum, and this encouraged the use of this bound within a branch-and-bound framework. In principle, according to worst-case analysis, the lower bound can be not so close to the optimum, namely the following holds

Theorem 2 *([6], [10]) For every permutation π, $b(\pi) - c(\pi) \geq \frac{2}{3}d(\pi)$. Moreover, there exist permutations for which $b(\pi) - c(\pi) = \frac{2}{3}d(\pi)$.*

From the description of the B&P algorithm in the next section, it will be clear that the fact that $d(\pi) > b(\pi) - c(\pi)$ would imply that at least one leaf node in the branch-decision tree must be explored in order to solve the instance. More precisely, each leaf node corresponds to a different ACD solution, and one has to explore *all* leaves corresponding to solutions of value $\overline{c}(\pi) > b(\pi) - d(\pi)$, which is clearly not desirable in a B&P algorithm. Therefore, in order to believe that the approach described in this paper and former ones is suited for the problem, one would like to be sure that $d(\pi) > b(\pi) - c(\pi)$ only for a very small fraction of the permutations. This is somehow proved in [7]

Theorem 3 *([7]) The probability that $d(\pi) > b(\pi) - c(\pi)$ for a random permutation π with n elements is $\Theta(1/n^5)$.*

From a complexity point of view, all the negative results for SBR apply to ACD as well [6, 4]. On the other hand, from a practical point of view, the great advantage of dealing with ACD instead of SBR is the fact that the former has a strong, natural ILP formulation with one variable for each alternating-cycle of $G(\pi)$, which was proposed in [25, 8]. Let $\mathcal{C}$ denote the set of all the alternating cycles of $G(\pi)$, and for each $C \in \mathcal{C}$ introduce a binary variable x_C. A natural ILP model is

$$\max \sum_{C \in \mathcal{C}} x_C \tag{1}$$

subject to

$$\sum_{C \ni e} x_C \leq 1, \qquad e \in E, \tag{2}$$

$$x_C \in \{0, 1\}, \qquad C \in \mathcal{C}. \tag{3}$$

Constraints (2) ensure that each edge in E is contained in at most one cycle C such that $x_C = 1$. As the removal of edges in an alternating cycle in a breakpoint graph yields a graph in which each node is incident with the same number of black and grey edges, any maximal solution of (1)-(3) will satisfy all constraints (2) at equality. Therefore, in an optimal solution x^* to (1)-(3), the set of cycles C such that $x^*_C = 1$ defines an optimal ACD solution.

A valid upper bound $\lfloor c^*(\pi) \rfloor$ on $c(\pi)$, and hence a valid lower bound $\lceil b(\pi) - c^*(\pi) \rceil$ on $d(\pi)$, is given by the optimal solution value $c^*(\pi)$ of the *LP relaxation* of (1)-(3), obtained by replacing constraints (3) with

$$x_C \geq 0, \qquad C \in \mathcal{C}. \tag{4}$$

(Notice that in the model defined by (1), (2) and (4), $x_C \leq 1$ for $C \in \mathcal{C}$ is trivially implied.) The dual of LP (1), (2) and (4) reads

$$\min \sum_{e \in E} u_e \tag{5}$$

subject to

$$\sum_{e \in C} u_e \geq 1, \qquad C \in \mathcal{C}, \tag{6}$$

$$u_e \geq 0, \qquad e \in E. \tag{7}$$

Solving the LP relaxation (1), (2) and (4) amounts to solving an LP having $|E| = O(n)$ constraints, and $|\mathcal{C}| = O(2^n)$ variables, i.e. a possibly huge number of variables. Due to a fundamental result of Grötschel, Lovàsz and Schrijver [17] based on Khachian's ellipsoid algorithm [23], LP (1), (2) and (4) can be solved in polynomial time (in n) provided the following *column generation problem* is efficiently solvable: Given an $|E|$-dimensional vector u^*, test if u^* is *dual feasible*, i.e. if it satisfies (6) and (7). From a primal point of view, one would like to find a variable x_C having positive reduced-cost with respect to the dual vector u^*. Restated in combinatorial terms, the problem calls for finding an alternating cycle $C \in \mathcal{C}$ such that

$$\sum_{e \in C} u^*_e < 1, \tag{8}$$

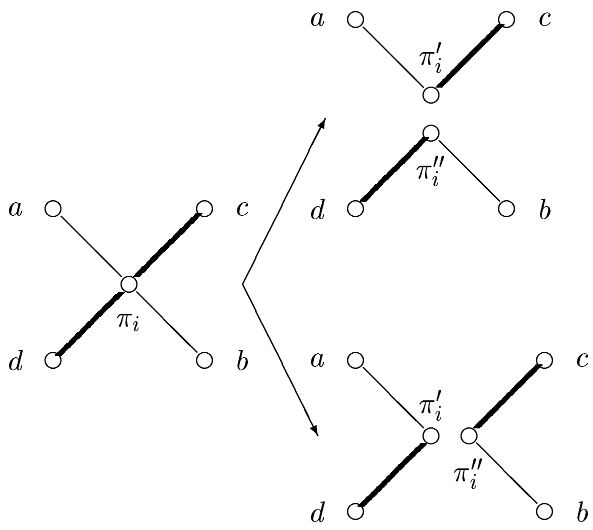

Figure 2: The two possible splittings of singleton π_i.

or proving that none exists. Reference [8] describes a polynomial-time algorithm for this problem, proving the following statement.

Theorem 4 *([8]) LP (1), (2) and (4) can be solved in polynomial time.*

In practice, in order to solve the LP relaxation, one starts from an LP with a restricted subset of the variables (implicitly fixing the other variables to 0). Then, one iteratively solves the current LP, tests if there are variables with positive reduced cost with respect to the current dual solution u^*, and, if this is the case, adds them to the current LP. The procedure terminates when there are no variables with positive reduced cost.

The algorithm presented in [8] to solve the column generation problem computes up to $n+1$ min-cost perfect matching problems in a suitably-defined (nonbipartite) graph. As this turns out to be computationally extremely expensive in practice, within the approach presented here we will solve a "relaxed" column generation problem by bipartite matching, as illustrated in Section 4.

3 Branch-and-Price for SBR from Branch-and-Price for ACD

A B&P algorithm for ACD follows immediately from formulation (1)-(3), as discussed below. In particular, a main issue to be addressed in B&P algorithms is the definition of a branching rule which preserves the structure of the column generation problem in the nodes of the branch-decision tree (see [2]). For the specific case of ACD, such a branching rule appears naturally if one thinks about the problem in the following terms. An ACD solution is uniquely defined by considering each node of degree 4 of $G(\pi)$ and stating that either y_1, b_1 and y_2, b_2 or y_1, b_2 and y_2, b_1 are consecutive edge pairs in an alternating cycle of the solution, where y_1, y_2 are the grey edges and b_1, b_2 the black edges incident with the node. This observation is formalized below by introducing the notion of splitting of a singleton.

Given a singleton π_i of π, associated with a node of degree 4 in $G(\pi)$, the *splitting* of π_i corresponds to replacing node π_i by two nodes π_i' and π_i'', replacing the two grey edges incident with π_i, say (π_i, a) and (π_i, b), by gray edges (π_i', a) and (π_i'', b), and replacing the two black edges incident with π_i, say (π_i, c) and (π_i, d), either by black edges (π_i', c) and (π_i'', d) or by black edges (π_i', d) and (π_i'', c). Note that the splitting of node π_i removes this node and introduces two new nodes π_i' and π_i'': We will say that these two nodes are *associated* with π_i. For an unsplit node π_i, the unique node *associated* with π_i will be π_i itself. Figure 2 illustrates the splitting operation.

Clearly, there are two possible splittings for each singleton, and the graph obtained by splitting one or more singletons defines an ACD instance where original nodes of degree 4 have been replaced by pairs of nodes of degree 2, and whose optimal solution is not better than that of the original instance. Moreover, after having split all the singletons, the nodes in the resulting graph have degree 2, i.e. the graph is a collection of alternating cycles and the associated ACD instance has a unique solution.

By the above discussion, ACD can be stated as follows: Given a breakpoint graph, find a splitting for its singletons so as to maximize the number of alternating cycles in the resulting graph. Based on this easy observation, the natural branching rule that can be used in a B&P algorithm corresponds to splitting a suitably-chosen singleton in the two possible ways. In Section 5 we will give the details about this choice, showing in particular that, if the optimal LP solution is fractional, one can always branch so as to make this LP solution infeasible for the LPs in the two subproblems generated. Clearly, this branching rule is suited for a B&P approach, as the subproblems generated are still ACDs, and hence column generation can be carried out in the same way as at the root node.

Modulo a few details, the branching rule above, along with the column generation procedure described in the next section, define an effective B&P algorithm for ACD. Now the point is how to turn this algorithm into an algorithm for SBR, which is the problem that we would like to solve. The main step in this direction is the very nice interpretation of the splitting of a singleton in terms of SBR. To this aim we have to define a relevant variant of SBR, called *Signed SBR* (SSBR).

A *signed* permutation σ is obtained by taking a permutation π and *signing* each element π_i, replacing π_i by either π_i or $-\pi_i$, i.e. $\sigma_i = \pm\pi_i$ for $i = 1, \ldots, n$. The reversal operation applied to σ has the effect of both inverting a subsequence from σ_i to σ_j (as in the unsigned case) and flipping the signs of the elements in the subsequence, replacing $(\sigma_1 \ \ldots \ \sigma_{i-1} \ \sigma_i \ \sigma_{i+1} \ \ldots \ \sigma_{j-1} \ \sigma_j \ \sigma_{j+1} \ \ldots \ \sigma_n)$ by $(\sigma_1 \ \ldots \ \sigma_{i-1} - \sigma_j \ - \sigma_{j-1} \ \ldots \ - \sigma_{i+1} \ - \sigma_i \ \sigma_{j+1} \ \ldots \ \sigma_n)$. Given a signed permutation σ, SSBR calls for a shortest sequence of reversals transforming σ into the (signed) identity permutation $(1 \ 2 \ \ldots \ n-1 \ n)$. SSBR is in principle closer to the real-world genome rearrangement problem, in that genes have an orientation which can be represented by signs. Nevertheless, this orientation is unknown in most cases, and this motivates the interest for SBR.

Given a permutation π, it is easy to see that there exists a signing of the elements of π yielding a signed permutation σ such that an optimal sequence of reversals for σ is also optimal for π (see also Remark 2 below). In view of the following discussion, there is no polynomial-time algorithm to find such a signing unless P = NP. A breakthrough result obtained by Hannenhalli and Pevzner [19] is that SSBR can be solved in polynomial time. The algorithm they proposed was later improved by Berman and Hannenhalli [3] and Kaplan, Shamir and Tarjan [22], leading to the following

Theorem 5 *([19], [3], [22]) SSBR can be solved in $O(n^2)$ time.*

In this context, we are interested in an intermediate problem between SBR and SSBR, hereafter called *Partially-Signed SBR* (PSSBR), in which only a fraction of the elements of the given permutation are signed, and the effect of a reversal is to flip the signs of the signed elements in the associated subsequence. PSSBR was studied in [20], by using some of the tools introduced in [19] for SSBR. In particular, we are interested in partially-signed permutations obtained from the original permutation π by signing (some of) its singletons. Let τ be such a permutation. First of all, note that

Remark 1 *Any sequence of reversals that defines a feasible PSSBR solution for τ also defines a feasible SBR solution for π.*

Remark 2 *For every permutation π, there exists a partially-signed permutation τ obtained by signing the singletons of π such that an optimal sequence of reversals for τ is also optimal for π.*

Proof. Consider an optimal sequence of reversals for π. For each singleton π_i of π, let $\tau_i := \pi_i$ if π_i is involved in an even number of reversals and $\tau_i := -\pi_i$ otherwise, where we say that an element is involved in a reversal if it is contained in the subsequence which is inverted by the reversal. The optimal sequence of reversals for π is a feasible, and therefore optimal, solution of PSSBR for τ. On the other hand, by Remark 1, this implies that any optimal PSSBR solution for τ is also an optimal SBR solution for π. □

Moreover, if all the singletons are signed, Hannenhalli and Pevzner [20] have shown that the corresponding PSSBR instance can be solved efficiently; by using the algorithm of [22] one may state their result as

Theorem 6 *([20],[22]) For every permutation π, the PSSBR instance defined by a partially-signed permutation τ obtained by signing all the singletons of π can be solved in $O(n^2)$ time.*

Accordingly, in principle, one gets an optimal solution of SBR for π by enumerating all the possible signings of the singletons of π and (efficiently) solving the associated PSSBR instance. This is the approach followed in [20].

The key point of our approach is that we can associate with a PSSBR instance, in which some of the singletons of π are signed, the breakpoint graph which is obtained from $G(\pi)$ by splitting the singletons which are signed. To be more specific, the breakpoint graph associated with a permutation τ obtained from π by signing some of its singletons is defined from the breakpoint graph $G(\pi)$ as follows. For each singleton π_i which is signed in τ, we split π_i by replacing π_i by π_i' and π_i''. Moreover, let (π_i, a) and (π_i, b) be the grey edges incident with π_i, where a is associated with $\pi_i - 1$ and b with $\pi_i + 1$, and (π_i, c) and (π_i, d) be the the black edges incident with π_i, where c is associated with π_{i-1} and d with π_{i+1}. We replace edges (π_i, a) and (π_i, b) by grey edges (π_i', a) and (π_i'', b). Furthermore, if $\tau_i = \pi_i$, we replace edges (π_i, c)

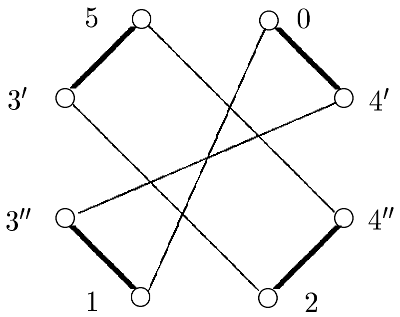

Figure 3: The breakpoint graph $G(\tau)$ associated with $\tau = (+4\ 2\ 1\ -3)$, where singletons 4 and 3 are signed.

and (π_i, d) by black edges (π'_i, c) and (π''_i, d); otherwise $\tau_i = -\pi_i$, and we replace edges (π_i, c) and (π_i, d) by black edges (π'_i, d) and (π''_i, c). We denote the graph obtained by $G(\tau)$. Figure 3 depicts the breakpoint graph $G(\tau)$ associated with $\tau = (+4\ 2\ 1\ -3)$.

According to the discussion above, the natural branching rule for ACD defined by splitting a singleton corresponds to signing the singleton in the original SBR instance in the two possible ways. Hence, in the aforementioned B&P algorithm, the leaf nodes of the branch-decision tree correspond to the partially-signed permutations obtained by signing all the singletons of π. Whereas the ACD solution for these instances is unique, and hence trivially determined, the solution of the corresponding PSSBR instances can be determined in $O(n^2)$ time by Theorem 6. Therefore, from Remark 2, an explicit complete exploration of this branch-decision tree solves in principle the original SBR instance to proven optimality. This is not a practical approach, in that one needs a (strong) lower bound to fathom a vast majority of the nodes of the branch-decision tree. As we point out in the following, this lower bound is given by the LP upper bound for ACD.

For a generic node of the branch-decision tree, let τ be the corresponding partially-signed permutation. Consider the ACD instance defined by $G(\tau)$, and let $c(\tau)$ be its optimal solution value. Moreover, let $d(\tau)$ be the optimal solution value of PSSBR for τ, and $b(\tau)$ $(= b(\pi))$ be the number of black edges of $G(\tau)$. By the same considerations leading to the proof of Theorem 1 one can show

Theorem 7 *([1], [20]) For every partially signed permutation τ (obtained from π by signing some singletons), $d(\tau) \geq b(\tau) - c(\tau)$.*

Hence, the LP relaxation of ACD, of value $c^*(\tau)$, yields a valid (and typically tight) lower bound $\lceil b(\tau) - c^*(\tau) \rceil$ on $d(\tau)$.

There is still a small detail that has to be taken into account in the definition of a B&P algorithm for SBR from the one for ACD. Consider a generic node of the branch-decision tree, corresponding to the partially signed permutation τ. If the LP solution is integer, then its value is $c(\tau)$, and the corresponding ACD instance is solved to optimality. Hence, in an algorithm for ACD one would not branch further. For SBR, if not all the singletons are signed and no feasible solution of value $b(\tau) - c(\tau)$ (or smaller) is known, one has to keep on branching

algorithm B&P SBR($\pi, d(\pi)$);
input: a permutation π of $\{1, \ldots, n\}$;
output: the optimal SBR solution value $d(\pi)$ for π;
begin
  $d(\pi) := \infty$; **comment**: upper bound on the optimal solution value;
  $Q := \{\pi\}$; **comment**: list of partially signed permutations to be considered;
  repeat
    let τ be a partially signed permutation in Q;
    $Q := Q \setminus \{\tau\}$;
    if all singletons are signed in τ **then begin**
      solve to optimality PSSBR for τ, letting $d(\tau)$ denote the optimal solution value;
      if $d(\pi) > d(\tau)$ **then** $d(\pi) := d(\tau)$;
    end
    else begin
      solve the LP relaxation of ACD for $G(\tau)$, letting $c^*(\tau)$ denote the optimal solution value;
      if $d(\pi) > \lceil b(\tau) - c^*(\tau) \rceil$ **then begin**
        comment: the optimal PSSBR solution value for τ may be better than the current $d(\pi)$ (otherwise the node is fathomed);
        generate two partially signed permutations τ_1 and τ_2 by giving a positive and a negative sign, respectively, to a singleton unsigned in τ;
        $Q := Q \cup \{\tau_1, \tau_2\}$;
      end;
    end;
  until $Q = \emptyset$;
end.

Figure 4: Overview of the B&P algorithm for SBR.

by signing the singletons unsigned so far. (Note that on one branch the LP solution will be unchanged.) Nevertheless, this typically does not happen, as in almost all cases one quickly finds a SBR solution of value $d(\tau) = b(\tau) - c(\tau)$ if an ACD solution of value $c(\tau)$ is available. (This empirical observation is theoretically justified by the results in [7].) Details on how to find such a solution in most cases, and how to branch otherwise, will be given in Section 5.

We summarize the main skeleton of our B&P algorithm for SBR in Figure 4. We omit the (straightforward) details about storing an optimal sequence of reversals. Note that many points are left unspecified, for instance the choice of the singleton to sign for branching and the selection of the next subproblem to consider. All these issues are addressed in Section 5. The computational bottleneck of the algorithm is the solution of the LP relaxations of ACD in each node of the branch-decision tree. In the next section, we will present an alternative LP relaxation which is in practice much faster to solve.

A crucial point for the effectiveness of the B&P algorithm is the tightness of the lower bound $b(\pi)-c(\pi)$ on $d(\pi)$. In particular, as already mentioned in the introduction, if $d(\pi) > b(\pi)-c(\pi)$ one has to consider explicitly all the leaf nodes in the branch-decision tree corresponding to ACD solutions of value $> b(\pi) - d(\pi)$. We mention once more that this never happened in our computational experiments, and Theorem 3 justifies this behavior.

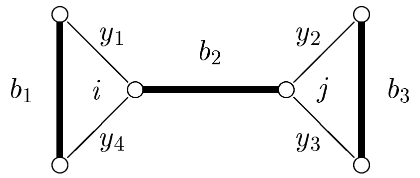

Figure 5: Pseudo alternating cycle $C = b_1, y_1, b_2, y_2, b_3, y_3, b_2, y_4$, where $\mu_{b_2,C} = 2$ and $\mu_{e,C} = 1$ for $e \in C \setminus \{b_2\}$.

4 A Much Faster to Solve LP Relaxation

As discussed in Section 2, the column generation problem for LP (1), (2) and (4) calls for an alternating cycle in $G(\pi)$ of weight < 1, where weights are given by the current values of the dual variables. The problem is the same in the nodes of the branch-decision tree, with the only difference that the graph $G(\pi)$ is replaced by $G(\tau)$. As already mentioned, this problem can be solved in polynomial time by min-cost matching, but the solution of these matching problems turns out to eat basically all of the overall computing time, even by using state-of-the-art codes for matching [16] and by trying a number of heuristic procedures to identify positive reduced-cost variables before resorting to the use of these codes, as explained in the next section. To overcome this serious drawback, we came up with an alternative LP relaxation, which is described in this section. This relaxation has a wider set of variables and hence is weaker than the original one. However, the associated column generation problem can be solved much faster by bipartite matching. Furthermore, we show that, even in the worst case, the lower bound provided by this new relaxation is not much worse than the previous one.

4.1 The new LP relaxation

A *surrogate alternating cycle* of $G(\pi)$ is a cycle whose edges are alternately black and grey, possibly with edge repetitions, that does not contain a strictly smaller surrogate alternating cycle. More precisely, a surrogate alternating cycle is a *minimal* sequence of edges $b_1, y_1, b_2, y_2, \dots, b_m, y_m$, such that:

(i) $b_i \in B$, $y_i \in Y$ for $i = 1, \dots, m$;

(ii) b_i and y_j have a common endpoint for $i = j = 1, \dots, m$ and for $i = j + 1$, $j = 1, \dots, m$;

(iii) b_i and b_{i+1} (resp. y_i and y_{i+1}) have no common endpoint for $i = 1, \dots, m$.

In other words, condition (iv) in the definition of alternating cycle has been removed. By *minimal* we mean that the sequence of edges does not contain another surrogate alternating cycle as a subsequence. Note that alternating cycles are also surrogate alternating cycles. We call *pseudo alternating cycle* a surrogate alternating cycle which is not an alternating cycle, i.e. such that $b_i = b_j$ or $y_i = y_j$ for some $1 \leq i < j \leq m$. An example of a pseudo alternating cycle is given in Figure 5. Let $\mathcal{S}$ denote the set of surrogate alternating cycles of $G(\pi)$ and $\mathcal{P} := \mathcal{S} \setminus \mathcal{C}$ the set of pseudo alternating cycles of $G(\pi)$.

The new LP relaxation is the counterpart of (1), (2) and (4), where the set $\mathcal{C}$ of alternating cycles is replaced by the set $\mathcal{S}$ of surrogate alternating cycles. In other words, the LP model reads

$$\max \sum_{C \in \mathcal{S}} x_C \tag{9}$$

subject to

$$\sum_{C \ni e} \mu_{e,C}\, x_C \leq 1, \qquad e \in E, \tag{10}$$

$$x_C \geq 0, \qquad C \in \mathcal{C}, \tag{11}$$

where, for each $C \in \mathcal{S}$ and $e \in C$, $\mu_{e,C}$ is the number of times that edge e appears in the sequence defining surrogate alternating cycle C. Obviously, $\mu_{e,C} = 1$ for each alternating cycle C and edge $e \in C$, and, by definition, $\mu_{e,C} \geq 2$ for at least one edge e in a pseudo alternating cycle C (in fact it is easy to show that $\mu_{e,C} \leq 2$). This yields the following

Remark 3 *In every integer solution of (9)-(11), $x_C = 0$ for all $C \in \mathcal{P}$.*

Therefore, by solving LP (9)-(11) instead of (1), (2) and (4), the bounds obtained may be weaker, but if the optimal LP solution is integer then we have an optimal ACD solution. In other words, with the additional restriction that the variables must be binary, (9)-(11) yields an alternative ILP formulation of ACD.

We conclude this subsection by some observations on the new LP relaxation that have interesting implications. First of all, one may note that an alternative LP relaxation is obtained by replacing inequalities (10) by equations

$$\sum_{C \ni e} \mu_{e,C}\, x_C = 1, \qquad e \in E. \tag{12}$$

As long as integrality constraints are imposed, having inequalities (10) or equations (12) is clearly the same, as discussed in Section 2. The next lemma shows that the same holds for the LP relaxations as well.

Lemma 1 *Given any feasible solution $\overline{x}$ of (9)-(11) that does not satisfy some of (12), there is a feasible solution x^* of (9), (12) and (11) such that $x^*_C \geq \overline{x}_C$ for all $C \in \mathcal{S}$ and $x^*_C > \overline{x}_C$ for at least one $C \in \mathcal{S}$.*

Proof. Given $\overline{x}$, a solution x^* with the desired properties can be constructed as follows. Consider a node i incident with a black edge for which the associated constraint (12) is not satisfied. This implies that $\sum_{C \in \mathcal{S}(i)} \overline{x}_C$ is strictly smaller than 1 if i has degree 2, and smaller than 2 if i has degree 4, where $\mathcal{S}(i)$ denotes the set of surrogate alternating cycles visiting node i. Hence, the node is also incident with at least one grey edge for which constraint (12) is not satisfied. In other words, given $\overline{x}$, the graph obtained from $G(\pi)$ by removing the edges for which (12) is satisfied is such that every node incident with a black edge is also incident with a grey edge and viceversa. This guarantees that this graph contains at least one surrogate alternating cycle C, and the associated variable $\overline{x}_C$ can be increased by a strictly positive value so as to keep all constraints (10) satisfied while satisfying at least one constraint (12) previously violated. Iterating this procedure yields the desired x^*. $\square$

An immediate consequence of this lemma is the following

Theorem 8 *The optimal solution value of LP relaxation (9)-(11) is equal to that of LP relaxation (9), (12) and (11).*

One would say that in practice LP relaxation (9)-(11) should be used, because the LPs should be easier for the LP solver and the dual variables are nonnegative, which may turn out to speed-up the convergence in the column generation process (see [2]) . On the other hand, using equality constraints has the advantage that one can *halve* the number of constraints in the LP, as we show now.

Theorem 9 *The LP relaxation defined by (9), (11) and the constraints (12) associated with black edges, namely*

$$\sum_{C \ni e} \mu_{e,C}\ x_C = 1, \qquad e \in B, \tag{13}$$

is equivalent to LP relaxation (9), (12) and (11), in the sense that the set of feasible solutions of the two LPs coincide.

Proof. Consider a solution x^* of (9), (13) and (11). We show that constraints (12) associated with grey edges are satisfied by x^*. Recall that there is no cycle in $G(\pi)$ formed by grey edges only, i.e. the grey edges form node-disjoint paths whose endpoints are nodes of degree 2 in $G(\pi)$ and whose intermediate nodes have degree 4. Consider one such path, say $y_1, \ldots, y_p$. We prove the claim by showing that constraints (12) associated with the edges in this path are satisfied. Let i be a node of degree 2 incident with edge y_1. Due to constraint (13) associated with the black edge incident with i, one has that $\sum_{C \in \mathcal{S}(i)} x^*_C = 1$, where $\mathcal{S}(i)$ denotes the set of surrogate alternating cycles visiting node i, implying that $\sum_{C \ni y_1} x^*_C = 1$, i.e. constraint (12) is satisfied for edge y_1. Now consider the node j incident with y_1 and y_2. As $\sum_{C \in \mathcal{S}(j)} x^*_C = 2$, due to constraints (13), and $\sum_{C \ni y_1} x^*_C = 1$, we have $\sum_{C \ni y_2} x^*_C = 1$. Iterating the argument shows that $\sum_{C \ni y_i} x^*_C = 1$ for $i = 1, \ldots, p$. □

Therefore, we work with LP (9), (13) and (11) within our implementation, as having half the number of constraints is far more advantageous than having inequalities instead of equations.

4.2 Column generation by bipartite matching

The dual of (9), (13) and (11) has the form

$$\min \sum_{e \in B} u_e \tag{14}$$

subject to

$$\sum_{e \in C} \mu_{e,C}\ u_e \geq 1, \qquad C \in \mathcal{S}, \tag{15}$$

where u_e, $e \in B$, is unconstrained in sign. The associated column generation problem requires finding, if any, a surrogate alternating cycle $C \in \mathcal{S}$ such that $\sum_{e \in C} \mu_{e,C}\ u^*_e < 1$ for a given u^* vector. Call $\sum_{e \in C} \mu_{e,C}\ u^*_e$ the *weight* of $C \in \mathcal{S}$. We next show how to solve this problem.

Construct the arc-weighted directed graph $D(\pi) = (V, A)$ from $G(\pi)$ and u^* as follows. $D(\pi)$ has the same node set V as $G(\pi)$ and, roughly speaking, each path consisting of a black and a grey edge in $G(\pi)$ is replaced by an arc in $D(\pi)$. Formally, for each node pair $i, j \in V$,

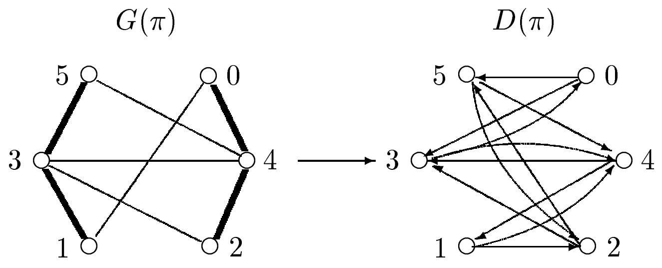

Figure 6: Graph $G(\pi)$ and the associated graph $D(\pi)$.

$D(\pi)$ has an arc $(i,j) \in A$ if there exists $k \in V$ such that $(i,k) \in B$ and $(k,j) \in Y$. The arc weight for (i,j) is given by $u^*_{(i,k)}$. If there exist two such k, we consider the one leading to the minimum weight of arc (i,j). Figure 6 shows the graph $D(\pi)$ associated with graph $G(\pi)$ in Figure 1. Let $\mathcal{A}$ denote the set of *simple* (i.e. without node repetitions) directed cycles of $D(\pi)$, called *dicycles* in the following.

Theorem 10 *Each dicycle of $D(\pi)$ corresponds to a surrogate alternating cycle of $G(\pi)$ of the same weight. Viceversa, each surrogate alternating cycle of $G(\pi)$ corresponds to a dicycle of $D(\pi)$ whose weight is not larger.*

Proof. Consider a dicycle of $D(\pi)$, corresponding to arc sequence $a_1, a_2, \ldots, a_m = (i_1, i_2), (i_2, i_3), \ldots, (i_m, i_1)$. For $h = 1, \ldots, m$, let j_h be a node such that black edge (i_h, j_h) and grey edge (j_h, i_{h+1}) are in $G(\pi)$. (Note that there may exist one or two such j_h; in the latter case we choose the one for which $u^*_{(i_h,j_h)}$ is smaller.) It is then immediate to see that the edge sequence $b_1, y_1, \ldots, b_m, y_m = (i_1, j_1), (j_1, i_2), \ldots, (i_m, j_m), (j_m, i_1)$ is a surrogate alternating cycle C of $G(\pi)$. In particular, suppose the sequence is not minimal, i.e. there exists a subsequence having either the form $b_s, y_s, \ldots, b_t, y_t = (i_s, j_s), (j_s, i_{s+1}), \ldots, (i_{t-1}, j_{t-1}), (j_{t-1}, i_s)$ or the form $y_s, b_{s+1}, \ldots, y_t, b_{t+1} = (j_s, i_{s+1}), (i_{s+1}, j_{s+1}), \ldots, (j_t, i_{t+1}), (i_{t+1}, j_s)$ which is a surrogate alternating cycle. But then the arc sequence defined by $(i_s, i_{s+1}), \ldots, (i_t, i_{t+1})$ in the first case and $(i_{s+1}, i_{s+2}), \ldots, (i_{t+1}, i_{s+1})$ in the second is a subsequence of $a_1, \ldots, a_m$ corresponding to a dicycle of $D(\pi)$, implying that $a_1, \ldots, a_m$ is not a dicycle of $D(\pi)$. The weight of the dicycle of $D(\pi)$ and that of C coincide.

Conversely, consider a surrogate alternating cycle C of $G(\pi)$, corresponding to edge sequence $b_1, y_1, \ldots, b_m, y_m = (i_1, j_1), (j_1, i_2), \ldots, (i_m, j_m), (j_m, i_1)$. The minimality requirement guarantees that $i_h \neq i_k$ for $1 \leq h < k \leq m$. Hence, the corresponding sequence of arcs $(i_1, i_2), \ldots, (i_m, i_1)$ is a dicycle of $D(\pi)$, whose weight is not larger than that of C by definition of $D(\pi)$. □

Corollary 1 *For a given u^*, $G(\pi)$ contains a surrogate alternating cycle $C \in \mathcal{S}$ of weight < 1 if and only if $D(\pi)$ contains a dicycle of weight < 1.*

We now describe how we check the existence of dicycles having weight < 1 in $D(\pi)$. First of all, we introduce *loops* in $D(\pi)$, i.e. arcs of the form (i,i) for all $i \in V$. The weight of the loops is initially set to 0, and a first *Assignment Problem* (AP) is solved on the distance matrix of $D(\pi)$, checking the existence of negative-weight dicycles in $D(\pi)$. The AP solution corresponds to a set of dicycles and loops of $D(\pi)$ such that each node is visited by exactly one dicycle or loop in the set, and the sum of the weights of the arcs in the dicycles and loops is a minimum. The problem coincides with the min-cost perfect matching problem on the complete bipartite graph having $|V|$ nodes on each shore of the bipartition, and where the cost of edge (i, j) is equal to the weight of arc (i, j) if $(i, j) \in A$, to $+\infty$ otherwise (see e.g. [16]).

If negative-weight dicycles exist, some will be present in the AP solution when loop weights are set to 0 and we are done, by Theorem 10, as the corresponding surrogate alternating cycles of $G(\pi)$ have negative weight.

If no negative-weight dicycle exists, the solution of the first AP corresponds to all the loops. Note that, provided an ACD solution is contained in the set of variables in the current LP, the "interesting" surrogate alternating cycles visit at least one node of degree 4, as possible surrogate alternating cycles visiting only nodes of degree 2 are already contained in the variable set. Therefore, we consider each node i of degree 4 in $G(\pi)$, in turn, set to $+\infty$ the weight of the corresponding loop in $D(\pi)$, and solve the AP for the new distance matrix. The new AP solution is formed by a minimum-weight dicycle in $D(\pi)$ visiting node i, and by loops. Hence, we get the min-weight surrogate alternating cycle visiting node i in $G(\pi)$. This latter AP need not be solved from scratch, since by starting from the solution of the first AP (with all loop weights equal to 0), the computation of an augmenting path is sufficient (see [16] for details). Trying all the nodes of degree 4 in $G(\pi)$ guarantees finding, if any, a dicycle of $D(\pi)$ (and a surrogate alternating cycle of $G(\pi)$) of weight < 1, thus solving the column generation problem.

In Figure 7 we give a formal, pseudo-code description of the procedure above. By using efficient implementations of the Hungarian method (see e.g. [16]), one can solve the initial AP in time $O(|V||A| \log |V|)$, and each of the other APs by just one augmentation step, in time $O(|A| \log |V|)$. Noting that $|A| = O(|V|) = O(n)$ due to the structure of $D(\pi)$, where in particular every $i \in V$ has at most 4 ingoing and outgoing arcs, one has the following

Theorem 11 *The column generation problem for LP (9), (13) and (11) can be solved in $O(n^2 \log n)$ time.*

Nevertheless, it is not the theoretical worst-case complexity but the much smaller time required in practice that determines the success of the procedure above with respect to a procedure to solve the column generation problem for LP (1), (2) and (4), based on the solution of a series of min-cost perfect matching problems on a nonbipartite graph [8].

4.3 Worst-case comparison of the old and new lower bounds

We now present a theoretical comparison of the lower bounds obtained by solving the old and new LP relaxations, motivated by the fact that these bounds turn out to be basically the same in practice. Let $c^*(\pi)$ denote the optimal solution value of LP (1), (2) and (4), and $\tilde{c}(\pi)$ the optimal solution value of LP (9), (13) and (11). Clearly, $c^*(\pi) \leq \tilde{c}(\pi)$, i.e. $\lceil b(\pi) - c^*(\pi) \rceil$ is a better lower bound on $d(\pi)$ than $\lceil b(\pi) - \tilde{c}(\pi) \rceil$. As we will see, there are examples in which $c^*(\pi) = 1$ and $\tilde{c}(\pi) = \Theta(n)$ for arbitrarily large n, implying that the worst-case ratio between $\tilde{c}(\pi)$ and $c^*(\pi)$ can be very bad. Anyway, we will show in the following that the ratio between the lower bounds on SBR is bounded by $\frac{3}{4}$, giving a partial theoretical explanation of empirical evidence.

procedure ColGen($G(\pi), u^*, \mathcal{F}$);
input: a breakpoint graph $G(\pi)$ and a vector u^* of weights for the edges of $G(\pi)$;
output: a set $\mathcal{F}$ of surrogate alternating cycles of $G(\pi)$ of weight < 1 — if $\mathcal{F} = \emptyset$ then no such cycle exists;
begin
  define $D(\pi) = (V, A)$ from $G(\pi)$ and u^*, letting w_a denote the weight of each arc $a \in A$;
  for $i := 1$ **to** $|V|$ **do** $w_{(i,i)} := 0$;
  solve the AP associated with $D(\pi)$ letting v denote the optimal solution value;
  comment: $v \leq 0$ and all the possible dicycles in the solution have weight ≤ 0;
  if the AP solution contains some dicycle **then**
    let $\mathcal{F} := \{C_1, \ldots, C_p\}$, where $C_1, \ldots, C_p$ are the surrogate alternating cycles of $G(\pi)$ corresponding to the dicycles in the AP solution
  else begin
    comment: all the dicycles of $D(\pi)$ have weight ≥ 0;
    $\mathcal{F} := \emptyset$;
    for each singleton i of π **do begin**
      $w_{(i,i)} := +\infty$;
      solve the AP associated with $D(\pi)$ letting v denote the optimal solution value;
      comment: the only dicycle of the AP solution visits node i and has weight v;
      if $v < 1$ **then** $\mathcal{F} := \mathcal{F} \cup \{C\}$, where C is the surrogate alternating cycle of $G(\pi)$ corresponding to the only dicycle in the AP solution;
      $w_{(i,i)} := 0$;
    end
  end
end;

Figure 7: Column generation procedure.

Lemma 2 *For every pseudo alternating cycle C of $G(\pi)$,*

$$\sum_{e \in B \cap C} \mu_{e,C} \geq 4,$$

i.e. every pseudo alternating cycle of $G(\pi)$ contains at least 4 black edges, counting each edge with its multiplicity within the cycle.

Proof. Let $b_1, y_1, b_2, y_2, \ldots, b_m, y_m$ be the edge sequence defining a surrogate alternating cycle. If $m = 2$, we show that $b_1 \neq b_2$ and $y_1 \neq y_2$. Suppose indeed $b_1 = b_2 = (i,j)$. This implies $y_1 = (i,j)$, which is impossible as $G(\pi)$ has no parallel edges. Similarly, for the case $m = 3$, the same reasoning shows that $b_1 = b_2 = (i,j)$ would imply $y_1 = (i,j)$, $b_2 = b_3 = (i,j)$ would imply $y_2 = (i,j)$ and $b_3 = b_1 = (i,j)$ would imply $y_3 = (i,j)$. Hence, if $m \leq 3$, the surrogate alternating cycle is necessarily an alternating cycle. □

Theorem 12 *For every permutation π,*

$$b(\pi) - \tilde{c}(\pi) \geq \frac{3}{4}(b(\pi) - c^*(\pi)).$$

Moreover, there exist permutations for which $b(\pi) - \tilde{c}(\pi)$ is arbitrarily close to $\frac{3}{4}(b(\pi) - c^*(\pi))$.

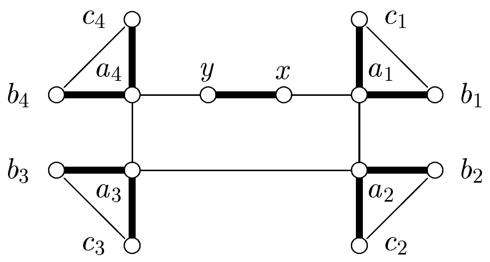

Figure 8: Graph G_4.

Proof. Let $\tilde{c}(\pi) = \tilde{c}_{\mathcal{C}} + \tilde{c}_{\mathcal{P}}$, where $\tilde{c}_{\mathcal{C}}$ is the contribution to objective function (9) of the variables corresponding to alternating cycles of $G(\pi)$, whereas $\tilde{c}_{\mathcal{P}}$ is the contribution of the variables corresponding to pseudo alternating cycles of $G(\pi)$. Note that

$$\tilde{c}(\pi) = \tilde{c}_{\mathcal{C}} + \tilde{c}_{\mathcal{P}} \geq c^*(\pi) \geq \tilde{c}_{\mathcal{C}}. \tag{16}$$

Recall that $b(\pi)$ is the number of black edges of $G(\pi)$, and note that every alternating cycle of $G(\pi)$ contains at least two black edges, as $G(\pi)$ has no parallel edges, whereas by Lemma 2 every pseudo alternating cycle of $G(\pi)$ contains at least 4 black edges. Adding up constraints (13) we get

$$b(\pi) = \sum_{e \in B} \sum_{C \ni e} \mu_{e,C}\; x_C = \sum_{C \in \mathcal{S}} \beta_C\; x_C = \sum_{C \in \mathcal{C}} \beta_C\; x_C + \sum_{C \in \mathcal{P}} \beta_C\; x_C,$$

where $\beta_C := \sum_{e \in B \cap C} \mu_{e,C}$ for $C \in \mathcal{S}$. As $\beta_C \geq 2$ if $C \in \mathcal{C}$ and $\beta_C \geq 4$ if $C \in \mathcal{P}$, we get

$$b(\pi) = \sum_{C \in \mathcal{C}} \beta_C x_C + \sum_{C \in \mathcal{P}} \beta_C x_C \geq \sum_{C \in \mathcal{C}} 2x_C + \sum_{C \in \mathcal{P}} 4x_C$$

which implies that

$$\sum_{C \in \mathcal{P}} x_C \leq \frac{b(\pi)}{4} - \frac{1}{2} \sum_{C \in \mathcal{C}} x_C.$$

Hence, for the solution of (9), (13) and (11), say $\tilde{x}$,

$$\tilde{c}_{\mathcal{P}} = \sum_{C \in \mathcal{P}} \tilde{x}_C \leq \frac{b(\pi)}{4} - \frac{1}{2} \sum_{C \in \mathcal{C}} \tilde{x}_C = \frac{b(\pi)}{4} - \frac{\tilde{c}_{\mathcal{C}}}{2}. \tag{17}$$

Using (17) and the last inequality in (16), one gets

$$\frac{b(\pi) - \tilde{c}(\pi)}{b(\pi) - c^*(\pi)} \geq \frac{b(\pi) - \tilde{c}_{\mathcal{C}} - \frac{b(\pi)}{4} + \frac{\tilde{c}_{\mathcal{C}}}{2}}{b(\pi) - \tilde{c}_{\mathcal{C}}} \geq \frac{b(\pi) - \tilde{c}_{\mathcal{C}} - \frac{b(\pi)}{4} + \frac{\tilde{c}_{\mathcal{C}}}{4}}{b(\pi) - \tilde{c}_{\mathcal{C}}} = \frac{3}{4}.$$

To show that the $\frac{3}{4}$ bound is (asymptotically) tight, consider the following family of breakpoint graphs (see Figure 8 for an illustration). Given a positive integer m, graph G_m has $3m+2$ nodes; let the node set be $\{a_i, b_i, c_i : i = 1, \ldots, m\} \cup \{x, y\}$. Graph G_m contains grey edges $(x, a_1), (a_1, a_2), (a_2, a_3), \ldots, (a_{m-1}, a_m), (a_m, y)$, and black edge (x, y), which altogether form a "main" (chordless) cycle of length $m+2$. Moreover, for $i = 1, \ldots, m$, G_m contains black edges $(a_i, b_i), (c_i, a_i)$ and grey edge (b_i, c_i), forming a triangle with a node in common with the

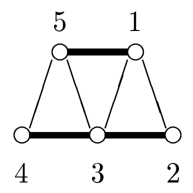

Figure 9: A reducible surrogate alternating cycle.

main cycle. It is easy to verify that G_m is the breakpoint graph associated with permutation $\pi = (3m\ 3m-3\ 3m-6\ \ldots\ 3i\ 3(i-1)\ \ldots\ 6\ 3\ 1\ 2\ 4\ 5\ \ldots\ 3i-2\ 3i-1\ \ldots\ 3m-2\ 3m-1)$; see [6] for a simple procedure to recognize breakpoint graphs and derive an associated permutation. Clearly $G(\pi)$ contains only one alternating cycle, defined by all the edges, and therefore $c(\pi) = c^*(\pi) = 1$. On the other hand, consider the solution of LP (9), (13) and (11) where the variables corresponding to the following pseudo alternating cycles have value $\frac{1}{2}$ (whereas the remaining variables have value 0):

- $(a_i, b_i), (b_i, c_i), (c_i, a_i), (a_i, a_{i+1}), (a_{i+1}, b_{i+1}), (b_{i+1}, c_{i+1}), (c_{i+1}, a_{i+1}), (a_{i+1}, a_i)$, for $i = 1, \ldots, m-1$;
- $(a_1, b_1), (b_1, c_1), (c_1, a_1), (a_1, x), (x, y), (y, a_m), (a_m, b_m), (b_m, c_m), (c_m, a_m), (a_m, y), (y, x), (x, a_1)$.

It is immediate to verify that this solution is feasible for (9), (13) and (11). Furthermore, the value of this solution is $\frac{m}{2}$, implying $\tilde{c}(\pi) = \tilde{c}_{\mathcal{P}} \geq \frac{m}{2}$ (it would be easy to show that $\tilde{c}(\pi) = \frac{m}{2}$). Finally, $b(\pi) = 2m + 1$. Hence,

$$\frac{b(\pi) - \tilde{c}(\pi)}{b(\pi) - c^*(\pi)} \leq \frac{2m + 1 - \frac{m}{2}}{2m} = \frac{3}{4} + \frac{1}{2m}$$

that gets arbitrarily close to $\frac{3}{4}$ as m gets to infinity. □

As already mentioned, this theorem gives a partial theoretical explanation of empirical evidence, namely that the old and new LP relaxations give essentially the same lower bounds in practice. A much more satisfactory explanation might probably be obtained by considering the *expected* (absolute or relative) gap between the two bounds for a (uniformly) random permutation, along the same line as the results presented in [7]. Anyway, such a result is likely to require an extremely complex analysis.

4.4 A characterization of "relevant" surrogate alternating cycles

We conclude this section with an analysis of the structure of the surrogate alternating cycles that may have the corresponding variable strictly positive in the optimal solution of LP (9), (13) and (11).

We call a surrogate alternating cycle *reducible* if its edge set contains another surrogate alternating cycle as a subset, *unreducible* otherwise. Note that the minimality assumption does not guarantee unreducibility; for instance in Figure 9 edge sequence $(5,1), (1,2), (2,3), (3,1), (1,5), (5,3), (3,4), (4,5)$ is minimal, and hence defines a surrogate alternating cycle, which is reducible because $(5,1),(1,2),(2,3),(3,5)$ is an alternating cycle. Note that alternating cycles are always unreducible. The following remark can be proved in the same way as Lemma 1.

Remark 4 *In any optimal solution $\tilde{x}$ of (9), (13) and (11), $\tilde{x}_C = 0$ for all reducible surrogate alternating cycles C.*

Hence, we can restrict our attention to unreducible surrogate alternating cycles, that are characterized by the following theorem. We give some definitions before stating the theorem. Given a subset C of the edges of $G(\pi)$, the subgraph *induced by C* is the one obtained by removing from $G(\pi)$ all the edges not in C and the nodes not incident with any edge of C. Let S be a connected subgraph of $G(\pi)$. A *black-grey cut node* is a node i such that both the removal of the black edges incident with i and the removal of the grey edges incident with i disconnects S. For instance, both node i in Figure 5 and node a_1 in Figure 8 are black-grey cut nodes. A *multisubset* C of a set E is a set $C \subseteq E$ that may contain multiple copies of some element $e \in E$. Let $\mu_{e,C}$ denote the multiplicity of element e in multisubset C ($\mu_{e,C} = 0$ if $e \notin C$).

Theorem 13 *A multisubset C of the edges of $G(\pi)$ is the edge set of an unreducible surrogate alternating cycle $C \in \mathcal{S}$ if and only if the subgraph induced by C is connected and, for each node i of the subgraph induced by C,*

(i) $\sum_{e\in\delta(i)\cap B} \mu_{e,C} = \sum_{e\in\delta(i)\cap Y} \mu_{e,C}$,

(ii) $\sum_{e\in\delta(i)} \mu_{e,C} \in \{2,4\}$,

(iii) if $\sum_{e\in\delta(i)} \mu_{e,C} = 4$, *then i is a black-grey cut node of the subgraph induced by C,*

where $\delta(i)$ denotes the set of edges in C incident with i.

Proof. First of all, letting $(i_1,i_2),(i_2,i_3),\ldots,(i_{m-1},i_m)$ be the edge sequence corresponding to a surrogate alternating cycle C, we show that the subgraph induced by C satisfies the requirement. The fact that this graph is connected is obvious, and the definition of surrogate alternating cycle implies (i), which in turn implies that $\sum_{e\in\delta(i)} \mu_{e,C}$ is even. We now show (ii) and (iii).

The fact that i belongs to the subgraph induced by C implies that $\sum_{e\in\delta(i)} \mu_{e,C} \geq 2$. Suppose $\sum_{e\in\delta(i)} \mu_{e,C} \geq 6$. We then have $i = i_j = i_k = i_l$ for some $1 \leq j < k < l \leq m$. Assuming $(i_{j-1},i_j) \in B$ and $(i_j,i_{j+1}) \in Y$, the minimality requirement implies $(i_{k-1},i_k) \in Y$ and $(i_k,i_{k+1}) \in B$, otherwise $(i_j,i_{j+1}),\ldots,(i_{k-1},i_k)$ would be a surrogate alternating cycle contained in C. The same reasoning implies $(i_{l-1},i_l) \in B$, $(i_l,i_{l+1}) \in Y$ and hence $(i_{j-1},i_j) \in Y$, a contradiction. The case in which $(i_{j-1},i_j) \in Y$ and $(i_j,i_{j+1}) \in B$ is perfectly analogous. This shows (ii).

We finally show (iii). Let $\sum_{e\in\delta(i)} \mu_{e,C} = 4$ and $i = i_j = i_k$ for some $1 \leq j < k \leq m$. We consider the case $(i_{j-1},i_j) \in B$ and $(i_j,i_{j+1}) \in Y$, implying $(i_{k-1},i_k) \in Y$ and $(i_k,i_{k+1}) \in B$ (as before, the other case is analogous). If (iii) is violated, then there exists a node h such that $h = i_s = i_t$ for $j < s < k$ and either $1 \leq t < j$ or $k < t \leq m$. Assume $(i_{s-1},i_s) \in B$ and $(i_s,i_{s+1}) \in Y$, implying $(i_{t-1},i_t) \in Y$ and $(i_t,i_{t+1}) \in B$ (the other cases are analogous). We have that $(i_j,i_{j+1}),\ldots,(i_{s-1},i_s),(i_t,i_{t-1}),\ldots,(i_{k+1},i_k)$ is a surrogate alternating cycle contained in C, contradicting the minimality assumption.

We conclude the proof by showing that any multisubset of edges C such that the subgraph induced by C satisfies the requirements is the edge set of a surrogate alternating cycle $C \in \mathcal{S}$. As the subgraph is connected and each node is incident with the same number of black and grey edges (counting edges with their multiplicity), one may define from the edge set a sequence $b_1, y_1, \ldots, b_m, y_m$ satisfying properties (i)-(iii) in the definition of surrogate alternating cycle. To this end, consider for instance any Eulerian walk on the edges of C where edges are alternately black and grey, after having replaced the edges e such that $\mu_{e,C} = 2$ by two parallel edges of the same color. We still have to show unreducibility, which implies also minimality. Suppose the surrogate alternating cycle is not reducible. Consider a proper subset of C defining a surrogate alternating cycle, say C'. Clearly, there exists a node i such that $\sum_{e \in \delta(i)} \mu_{e,C} = 4$ and $\sum_{e \in \delta(i)} \mu_{e,C'} = 2$. This means that C' visits only one black and one grey edge incident with i, contradicting the fact that i is a black-grey cut node in the subgraph induced by C. □

5 Implementation Issues

In this section we illustrate the other main features of our B&P algorithm. Some of them represent points that were left open in the previous sections, as the selection of the singleton to split for branching, whereas others are relevant improvements to the main scheme, as the diving heuristic.

5.1 Branching choice and subproblem selection

The criterion for selecting the next subproblem to consider is *best-first*. Namely, each time we branch by generating two partially signed permutations τ_1, τ_2 from the current partially signed permutation τ, we associate with τ_1 and τ_2 the LP lower bound computed for τ. Then, among the partially signed permutations still to consider, we select the one for which the associated lower bound is smallest.

We already mentioned in Section 3 that we branch by splitting singletons. The choice of the singleton to be split is based on the optimal solution x^* of LP (9), (13) and (11). We say that a surrogate alternating cycle C is *removed* by splitting a node if C has no counterpart in the graph obtained after the splitting, that C is *kept* otherwise. Let i be a still unsplit singleton of π. First of all, letting $\mathcal{P}_i$ denote the set of the pseudo alternating cycles visiting node i, note that all the cycles in $\mathcal{P}_i$ will be removed by splitting i in both ways. Let a, b be the two grey edges incident with i and c, d the black edges incident with i, and consider the set of alternating cycles (excluding the pseudo alternating cycles) that contain edge a and not edge b (clearly, some of these cycles will be removed by splitting i). This set is partitioned into the set $\mathcal{C}_c(i)$ of the cycles containing edge c and the set $\mathcal{C}_d(i)$ of cycles containing edge d. Splitting i removes the cycles in $\mathcal{C}_c(i)$ in one subproblem and the cycles in $\mathcal{C}_d(i)$ in the other. Accordingly, if both $\sum_{C \in \mathcal{C}_c(i) \cup \mathcal{P}_i} x^*_C$ and $\sum_{C \in \mathcal{C}_d(i) \cup \mathcal{P}_i} x^*_C$ are "large", we can hope for a lower bound increase for both subproblems, as we are fixing to 0 variables whose sum is "large".

Based on the above considerations, we branch by splitting the singleton i such that $\sum_{C \in \mathcal{C}_c(i) \cup \mathcal{C}_d(i) \cup \mathcal{P}(i)} x^*_C$ is largest, breaking ties by in favour of the singleton i for which the absolute value of the difference $\sum_{C \in \mathcal{C}_c(i)} x^*_C - \sum_{C \in \mathcal{C}_d(i)} x^*_C$ is smallest.

Note that, in principle, x^* may be integer, of value $c(\tau)$, and we may have to branch as an SBR solution of value $b(\tau) - c(\tau)$ is not available (even if this typically does not happen in practice). In this case, we branch by splitting an arbitrary singleton.

5.2 Diving heuristic

In practice, finding a near-optimal solution of ACD (and SBR) seems to be a hard task, probably due to the very small number of these solutions. In our original implementation, at every node of the branch-decision tree we computed an ACD solution (and an associated SBR one) by signing the singletons in a greedy way based on the LP solution, as we describe in the next subsection. For large-size instances, when very few singletons are signed, the LP solution tends to be widely fractional, and the associated greedy solution is typically not very good. Therefore, for these instances, although the gap between the lower bound at the root node and the optimal solution value may be very small (0 in some cases), the B&P algorithm may require a very long time simply because it takes a very long time to find an optimal solution. Although the greedy heuristic remains active at every node of the tree, as it takes a very small time compared to the LP solution and sometimes it yields good solutions, we decided to apply the following more time consuming heuristic every K nodes of the branch-decision tree, where K is set to $n/4$ in our implementation.

As is common to many ILPs, when one derives an integer ACD solution by rounding a fractional LP solution, the main observation is the following. The variables whose LP value is higher are likely to be 1 in an optimal integer solution. Nevertheless, after fixing to 1 some of these variables, the LP value of the remaining ones does not give any more a reliable indication of their likelihood to be 1 in the optimal integer solution of the subproblem obtained. In practice, it is much better to solve again the LP after some variables have been fixed to 1, as the new LP solution will give a much more reliable indication than the old one. This typical observation inspires a classical *diving* heuristic in which one solves the LP, obtaining solution x^*, fixes to 1 *all* variables x_C with $x^*_C = 1$ as well as the fractional variable *closest to 1* corresponding to an alternating cycle (i.e. not to a pseudo alternating cycle), solves again the LP, and so on, until an integer solution is found. Note that fixing a variable x_C to 1 corresponds to splitting all the singletons visited by cycle C so as to keep C. Hence, the subproblem obtained after fixing one or more variables to 1 has the structure of the subproblems explored in the B&P algorithm, and the associated LP can be solved in the same way. This diving heuristic yields near-optimal solutions in most cases, as shown in the next section.

5.3 Greedy heuristic

As mentioned in the previous subsection, we apply a fast greedy heuristic to each subproblem explored in the B&P algorithm. Gven the optimal solution x^* of LP (9), (13) and (11), we consider the set $\mathcal{N} = \{C_1, \ldots, C_k\}$ of alternating cycles C such that $x^*_C > 0$. As long as $\mathcal{N} \neq \emptyset$, we iteratively consider the cycle $C \in \mathcal{N}$ for which x^*_C is maximum, split the singletons visited by C so as to keep C, and remove from $\mathcal{N}$ cycle C and all the cycles having some edges in common with C. If some singletons are still unsplit at the end of this procedure, we split them in an arbitrary way (typically this does not happen). This approach yields good ACD solutions when the number of unsplit singletons for the subproblem is not very large (say not larger than 100). This ACD solution is improved by considering each singleton, in turn, and testing if flipping only the sign assigned to this singleton yields a better solution, until no flipping yields an improvement (i.e. we have a local optimum). In order to get a heuristic SBR solution from an ACD one, we simply apply Theorem 6, solving to optimality the PSSBR instance obtained by splitting.

We note that the variant of the above procedure in which the cycle $C \in \mathcal{N}$ with highest x^*_C is kept with probability x^*_C, and simply removed from $\mathcal{N}$ otherwise, systematically yields worse results.

5.4 Greedy column generation

Although some orders of magnitude faster than the column generation approach based on general (nonbipatite) matching, the column generation approach of Section 4 remains one of the bottlenecks of the B&P algorithm, together with the solution of LPs. In order to avoid solving an AP at every iteration of the column generation process, we apply the following greedy heuristic to hopefully find dicycles of weight < 1 in $D(\pi)$.

We apply breadth-first search in $D(\pi) = (V, A)$ starting from a node i corresponding to a singleton of π and defining $S_0 := \{i\}$. We consider the set of nodes $S_1 = \{j \in V : (i, j) \in A\}$ and assign to each node $j \in S_1$ the label $\ell_j := w_{(i,j)}$, where w_a denotes the weight of arc $a \in A$. If $(j, i) \in A$ for some $j \in S_1$ and $\ell_j + w_{(j,i)} < 1$, we add the surrogate alternating cycle corresponding to dicycle $(i, j), (j, i)$ to our list $\mathcal{F}$ of surrogate alternating cycles of weight < 1. Then, we consider the set of nodes $S_2 := \{j \in V : (k, j) \in A \text{ for some } k \in S_1\} \setminus (S_0 \cup S_1)$, and assign to each node $j \in S_2$ the label $\ell_j := \ell_{k(j)} + w_{(k(j),j)}$, where $k(j)$ is an (arbitrarily chosen) node in S_1 such that $(k(j), j) \in A$. If $(j, i) \in A$ for some $j \in S_2$ and $\ell_j + w_{(j,i)} < 1$, we add to $\mathcal{F}$ the surrogate alternating cycle corresponding to dicycle $(i, k(j)), (k(j), j), (j, i)$ to $\mathcal{F}$. The procedure is iterated defining sets $S_3, S_4, \ldots$, until all nodes have been labeled.

The procedure above is applied starting with each singleton, checking if a cycle is already contained in $\mathcal{F}$ before adding it to $\mathcal{F}$, and terminating if the number of cycles in $\mathcal{F}$ is equal to W, set to $b(\pi)$ in our implementation, so as to avoid adding too many variables to the current LP.

5.5 Dynamic pricing out of variables in the LP

As anticipated in the previous subsection, besides column generation the bottleneck of the B&P algorithm is the solution of LPs by the LP solver. As it is often the case within column generation schemes, many of the variables that are generated turn out at some point to have value 0 in the LP, and keep this value until optimality is reached. As the LP solution time is affected by the number of variables even if these keep the value 0, and the number of variables generated throughout the procedure may be quite large, it would be convenient to remove these "useless" variables as soon as possible. Classical schemes [2] tend to remove LP variables whose reduced cost is above a certain threshold. This approach does not work in our case, as the high dual degeneracy has the effect that all the nonbasic variables have typically a very negative reduced cost, and hence for any reasonable threshold the effect would be to remove all the nonbasic variables. Instead, we simply keep in the LP the $R \times b(\pi)$ variables with largest reduced cost ($R = 2$ in our implementation), as well as all the variables with reduced cost $> C$ ($C = -0.01$ in our implementation), and remove the remaining variables.

6 Experimental Results

In this section we present the experimental results that we carried out to testify the effectiveness of our algorithm. We tested both real-world and randomly generated instances from the literature.

The section is organized as follows. We first illustrate the improvements obtained, respectively, by using LP relaxation (9), (13) and (11) in place of (1), (2) and (4), and by some of the algorithm enhancements described in Section 5. Then, we illustrate the results obtained by our algorithm on randomly generated instances, namely random permutations and permutations obtained by applying random reversals to the identity permutation. By *random permutation* we mean a permutation chosen with uniform probability among the $n!$ permutations with n elements. Anlogously, by *random reversal* we mean a reversal chosen with uniform probability. Finally, we report the optimal solution of real-world instances of the problem.

Our algorithm was coded in C and ran on a Digital Ultimate Workstation 500 MHz, which is approximately 2-3 times faster than a PC Pentium 350 MHz. The LP solver used is CPLEX 6.0. The solution of the Assignment Problems in the column generation phase was carried out using a C implementation of the Hungarian method (see e.g. [16]) along the same lines as the FORTRAN one by Carpaneto, Martello and Toth [9], adapted so as to take care of the sparsity of the cost matrix. The min-cost perfect matching code used for the solution of LP (1), (2) and (4) is a C implementation of an algorithm by Gabow [15] which is publicly available in the DIMACS web site (`www.dimacs.rutgers.edu`). We obtained analogous results by using the FORTRAN code by Derigs [11].

6.1 Improvements with the new LP relaxation

Table 1 reports the results obtained by solving instances associated with random permutations with our algorithm and with a variant that uses LP relaxation (1), (2) and (4), solving min-cost (nonbipartite) perfect matching problems in the column generation phase as illustrated in [8].

We solved instances up to size 150 as, for larger sizes, the variant could not solve any of the 10 test instances within the time limit, set to 1 hour. For each value of n, we report average values over 10 instances. For the variant (column *old LP*) we report the number of instances solved to proven optimality within the time limit (column *# solved*), the solution time (column *time*), the time spent in the solution of matching problems for column generation (*match time*), the average number of nodes considered in the branch-decision tree (*# nodes*) and the time spent in the root node (*root time*), mainly for the solution of the LP relaxation. For our algorithm (column *new LP*) we give the same information, in particular column *AP time* reports the time spent in the solution of assignment problems for column generation. Finally, we report the average ratios between the old and new LP relaxation values, before rounding up these values. (After rounding up, the lower bound values provided by these two LP relaxations turned out to be *always* the same for all the 60 instances tried.) Finally, we

n	old LP				new LP				$\frac{b(\pi)-\tilde{c}(pi)}{b(\pi)-c^*(pi)}$	speed-up
	# solved	time (match time)	# nodes	root time	# solved	time (AP time)	# nodes	root time		
25	10	0.04 (0.03)	1.0	0.04	10	0.01 (0.00)	1.1	0.01	1.000	7.0
50	10	2.74 (2.69)	1.3	2.27	10	0.07 (0.03)	1.0	0.04	0.999	50.4
75	10	69.39 (68.97)	4.3	24.02	10	0.52 (0.15)	2.8	0.25	1.000	77.3
100	10	618.83 (617.74)	6.2	160.89	10	2.78 (0.95)	13.9	0.55	1.000	187.8
125	9	1311.97 (1309.68)	4.7	346.15	10	4.83 (1.77)	9.1	1.26	1.000	134.6
150	3	3292.90 (3288.31)	6.9	654.48	10	22.50 (9.53)	32.8	2.23	1.000	293.5

Table 1: Results with the old and new LP relaxations. Average values over 10 instances.

n	compl		no DH			no GCG			no DPO		
	# solved	time	# solved	time	speed-up	# solved	time	speed-up	# solved	time	speed-up
25	10	0.01	10	0.01	1.75	10	0.01	2.00	10	0.02	2.25
50	10	0.07	10	0.07	1.08	10	0.24	3.69	10	0.24	3.72
75	10	0.52	10	0.42	0.80	10	2.07	3.95	10	2.02	3.86
100	10	2.78	10	2.18	0.78	10	4.76	1.71	10	4.84	1.74
125	10	4.83	10	10.63	2.20	10	23.05	4.78	10	22.80	4.72
150	10	22.50	10	52.39	2.33	10	35.85	1.59	10	35.53	1.58
175	10	400.93	9	613.16	1.53	10	338.12	0.84	10	338.27	0.84
200	9	956.55	5	2170.72	2.27	9	1166.78	1.22	9	1161.52	1.21

Table 2: Illustration of the algorithm enhancements. Average values over 10 instances.

give in column *speed-up* the speed-up factor achieved by using the new LP relaxation instead of the old one for the solution of the LP at the root node.

The table shows the terrific improvement in running time (from one to more than two orders of magnitude, depending on n) achieved by using the new LP relaxation. Furthermore, the ratio between the two bounds is almost equal to 1 in all cases, showing that the worst case ratio of 0.75 is overly pessimistic.

6.2 Improvements with the algorithm enhancements

In Table 2 we illustrate the improvements obtained by some of the enhancements described in Section 5. In particular, we compare the complete version of our algorithm, whose running time is reported in column *compl*, with versions in which, respectively, the diving heuristic was deactivated (column *no DH*), the greedy column generation procedure was deactivated (*no GCG*), and the dynamic pricing out of the variables was deactivated (*no DPO*). We solved instances associated with random permutations with up to 200 elements imposing a time limit of 1 hour. For each version, we report the number of instances solved out of 10 within the time limit (*# solved*), the running time (*time*) and the speed-up factor achieved by the complete version with respect to the version in which the enhancement was deactivated (*speed-up*).

The table shows that the diving heuristic starts to be very effective when n gets larger than 100, whereas the greedy column generation and the dynamic pricing out of variables are mainly helpful for n smaller than 150.

n	# singl	# rev	# sol	time	root time	LP time	AP time	# nodes	root gap
25	19.5 (23)	16.2 (18)	10	0.01 (0.02)	0.01 (0.02)	0.00 (50.0%)	0.00 (25.0%)	1.1 (2)	0.0% (0.0%)
50	42.1 (48)	35.0 (37)	10	0.07 (0.12)	0.04 (0.08)	0.03 (46.2%)	0.03 (43.6%)	1.0 (1)	0.0% (0.0%)
75	70.6 (75)	56.1 (58)	10	0.52 (1.37)	0.25 (0.32)	0.26 (49.7%)	0.15 (29.6%)	2.8 (16)	0.2% (1.8%)
100	95.1 (100)	76.1 (78)	10	2.78 (9.40)	0.55 (0.63)	1.17 (42.3%)	0.95 (34.3%)	13.9 (64)	0.3% (1.4%)
125	120.8 (125)	96.6 (98)	10	4.83 (12.52)	1.26 (1.47)	1.96 (40.6%)	1.77 (36.7%)	9.1 (34)	0.1% (1.0%)
150	144.1 (148)	116.6 (118)	10	22.50 (84.83)	2.23 (2.80)	8.89 (39.5%)	9.53 (42.3%)	32.8 (156)	0.7% (0.9%)
175	171.0 (173)	138.6 (141)	10	400.93 (1769.00)	3.33 (4.20)	142.82 (35.6%)	195.24 (48.7%)	506.7 (2341)	0.6% (1.4%)
200	196.8 (200)	159.1 (160)	9	956.55 (3600.00)	5.18 (7.13)	332.96 (34.8%)	479.55 (50.1%)	826.0 (3306)	0.6% (1.2%)

Table 3: Exact solution of random permutations. Average (maximum) values over 10 instances.

6.3 Results for randomly generated instances

Table 3 reports results of our B&P algorithm for random permutations with n up to 200. For each value of n we ran our algorithm on 10 instances, and we report the average (maximum) number of singletons in the permutation (column *# singl*), the average (maximum) number of reversals in the optimal (or best found) solution (*# rev*), the average (maximum) running time (*time*), the average (maximum) time spent at the root node (*root time*), the average time (average percentage over total time) for the solution of LPs by CPLEX (*time*), the average time (average percentage over total time) for the solution of assignment problems in the column generation phase, the average (maximum) number of nodes explored in the branch-decision tree (*# nodes*), the average (maximum) percentage gap between the optimal (or best found) solution value and the lower bound at the root node (*root gap*). This gap is computed as $\frac{d(\pi)-\lceil b(\pi)-\tilde{c}(\pi)\rceil}{d(\pi)}$.

These results show a considerable improvement with respect to the previous algorithms by Kececioglu and Sankoff [25] and by the authors [8], that could solve instances with n up to 40 and up to 100, respectively. Instances with $n = 100$ are now solved within 2-3 seconds. Within a time limit of 1 hour, we can consistently solve to proven optimality instances with n up to 200. We note that real-world instances typically have a smaller number of elements (see also the next subsection), and hence we expect our algorithm to be able to solve them to proven optimality within small computing time.

From Table 3, one can note that the average computing time over the 10 instances is quite different from the maximum computing time. This is a typical behavior of enumerative algorithms for hard problems, namely there may be a large difference in the time required to solve two instances of the same size generated in the same way.

We also tried to tackle larger instances by applying our diving heuristic algorithm. The results are given in Table 4, where the columns have the same meaning as in Table 3, with the exception of column *final gap*, that reports the average (maximum) percentage gap between the solution found and the LP lower bound.

Table 4 shows that even for instances with $n = 1000$ the heuristic consistently finds solutions within 2% of the optimum. The results in Section 4 prove that the heuristic has polynomial time complexity. Moreover, the table indicates that the actual running time is stable for different instances (i.e. the maximum and the average values are quite close to each other). Empirically, we have estimated the running time in $\Theta(n^4)$. In Figure 10 we compare the graph of the

n	# singl	# rev	time	root time	LP time	AP time	final gap
100	95.1 (100)	76.5 (79)	1.10 (1.50)	0.55 (0.63)	0.47 (43.2%)	0.31 (28.6%)	0.8% (2.5%)
200	196.8 (200)	159.5 (160)	16.85 (19.42)	5.20 (7.15)	4.74 (28.1%)	7.21 (42.8%)	0.9% (1.9%)
300	295.6 (300)	245.3 (247)	88.52 (93.75)	17.99 (22.08)	20.28 (22.9%)	39.51 (44.6%)	1.6% (2.4%)
400	395.0 (400)	330.2 (333)	270.43 (287.80)	49.16 (53.37)	60.73 (22.5%)	123.18 (45.6%)	1.6% (2.1%)
500	496.0 (500)	417.5 (422)	649.30 (667.13)	109.72 (130.00)	143.32 (22.1%)	301.85 (46.5%)	1.9% (2.6%)
600	596.8 (600)	502.8 (505)	1304.63 (1367.80)	185.55 (218.48)	269.34 (20.6%)	610.05 (46.8%)	1.8% (2.2%)
700	696.0 (700)	590.1 (593)	2465.88 (2632.55)	365.48 (461.97)	528.60 (21.4%)	1137.33 (46.1%)	1.9% (2.2%)
800	796.8 (800)	678.0 (682)	4345.65 (4509.78)	617.87 (850.43)	930.59 (21.4%)	2000.19 (46.0%)	1.9% (2.2%)
900	894.8 (898)	764.1 (766)	7062.10 (7543.95)	911.46 (1217.90)	1481.19 (21.0%)	3223.11 (45.6%)	1.9% (2.2%)
1000	994.4 (998)	851.5 (856)	10746.59 (11775.48)	1453.78 (2313.35)	2334.56 (21.7%)	4927.98 (45.9%)	1.9% (2.2%)

Table 4: Heuristic solution of random permutations. Average (maximum) values over 10 instances.

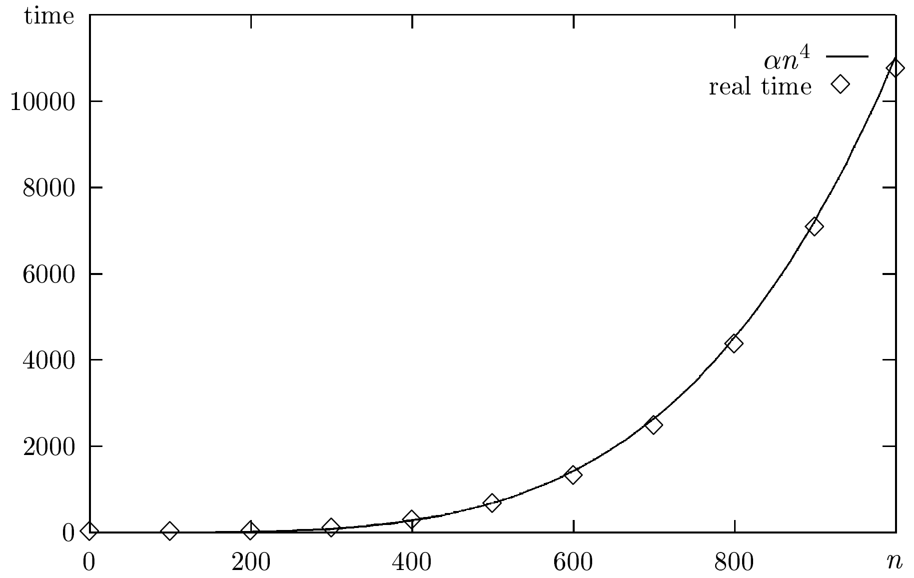

Figure 10: Average running time of diving heuristic versus function αn^4 with $\alpha = 1.1 \times 10^{-8}$.

function αn^4 and the plot of the actual average running times. Surprisingly, one can see that the two graphs are extremely close to each other. Note that the time spent for the first LP relaxation, given in column *root time*, is not as stable as the overall time for the heuristic.

Looking at Tables 3 and 4, one may note that the percentage LP time decreases with n while the percentage AP time increases with n. Overall, the sum of the two percentages seems to be asymptotically roughly equal to 85% for the exact algorithm and 67% for the heuristic.

We also solved instances obtained by using a generator by John Kececioglu that was made available at the time of the DIMACS Challenge. This generator starts from the identity permutation and applies random reversals to it.

We generated our instances by applying kn random reversals with $k = 1/8, 1/4, 1/2, 1, 2, 4, 8$. In Table 5 we report the results of our exact algorithm for n up to 200. In particular, for each value of n, the table gives the average solution value, the number of instances solved to optimality (within 1 hour) and the average computing time out of 10 instances.

n	$k = 1/8$			$k = 1/4$			$k = 1/2$			$k = 1$			$k = 2$			$k = 4$			$k = 8$		
	# rev	# sol	time	# rev	# sol	time	# rev	# sol	time	# rev	# sol	time	# rev	# sol	time	# rev	# sol	time	# rev	# sol	time
25	2.9	10	0.00	5.7	10	0.00	10.2	10	0.00	14.0	10	0.00	15.2	10	0.01	16.2	10	0.01	15.8	10	0.00
50	5.9	10	0.00	11.5	10	0.00	21.6	10	0.00	31.5	10	0.05	34.9	10	0.07	36.4	10	0.11	35.4	10	0.11
75	8.6	10	0.00	17.2	10	0.00	34.5	10	0.03	48.2	10	0.16	54.2	10	0.22	55.8	10	0.40	55.3	10	0.36
100	11.7	10	0.00	24.2	10	0.00	46.1	10	0.06	66.7	10	0.46	74.6	10	1.14	75.2	10	1.19	76.0	10	2.27
125	14.8	10	0.00	30.5	10	0.01	57.6	10	0.12	84.3	10	1.25	93.6	10	3.34	95.9	10	4.30	95.6	10	3.42
150	18.0	10	0.00	36.9	10	0.01	68.9	10	0.19	101.7	10	3.40	116.0	10	28.88	116.1	10	14.46	116.2	10	87.43
175	20.8	10	0.00	42.5	10	0.01	82.1	10	0.57	119.5	10	24.34	135.8	10	642.94	137.4	10	269.69	138.1	10	647.76
200	24.7	10	0.00	49.2	10	0.01	94.9	10	1.18	137.2	10	58.24	156.1	10	1680.36	158.4	9	694.11	158.0	8	1289.15

Table 5: Exact solution of permutations obtained by applying a sequence of random reversals to the identity. Average values over 10 instances.

instance	n	# singl	# rev	time	root time	LP time	AP time	# nodes	root gap
ignoring signs	138	133	106	10.58	0.45	5.11	4.36	18	1
partially signed	138	135	118	2.33	0.25	0.60	0.62	1	0

Table 6: Results for man and mouse genome instances.

The table shows that the permutations obtained from the generator are essentially random permutations for $k \geq 2$. For smaller k, the average reversal distance is smaller than for random permutations and the associated instances can be solved extremely fast. In particular, instances with $n = 200$ are solved within 1 minute for $k = 1$ and within 1 second for $k = 1/2$.

6.4 Results for real-world instances

We tested our algorithm on the largest real-world instances that we could find, which are obtained by comparing the genomes of men and mouse, and were kindly provided to us by Sridhar Hannenhalli [18]. The input is a partially signed permutation, meaning that the orientation of only part of the genes (namely, 47 out of 138) is known. From this input we derived two instances, the first by ignoring the signs of the elements and the second by considering the actual partially signed permutation. Table 6 illustrates the results of our algorithm on the two instances. The columns given in the table have the same meaning as the omonimous columns in Table 3. In particular, the minimum number of reversals is 106 if signs are ignored and 118 if they are taken into account. To our knowledge, our algorithm is the first one capable of finding a provably optimal solution for these two instances.

Moreover, we obtained from Mathieu Blanchette [5] a number of smaller size permutations associated with mitochondrial genomes. The size of these permutations is much smaller than the man-mouse one above, namely each permutation has 37 elements. We used our code to compute the distances between each pair of permutations, reported in Table 7, where we also report the scientific name of each species associated with the 20 genomes. The *overall* time for computing the whole table was 4.5 seconds, i.e., an average of about 0.025 seconds per genome pair. This shows that, the reversal distance for permutations of this size can be computed very fast, and therefore one may afford computing several times this distance within algorithms that try to reconstruct evolutionary trees (see e.g. [32]). Note that the table contains a few entries equal to 0, corresponding to pairs of species with the same arrangement of genes along the mitochondrial DNA.

Acknowledgments

The work of the first two authors was partially supported by MURST and CNR, Italy. We are grateful to Sridhar Hannenhalli and Mathieu Blanchette for having provided us with the real-world instances of the problem illustrated in Subsection 6.4. Finally, we thank Bob Carr, Matteo Fischetti, david Sankoff and Paolo Toth for helpful discussions on the subject.

References

[1] V. Bafna and P.A. Pevzner, "Genome Rearrangements and Sorting by Reversals", *SIAM Journal on Computing* 25 (1996) 272–289.

[2] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P.H. Vance, "Branch-and-Price: Column Generation for Solving Huge Integer Programs", *Operations Research* 46 (1998) 316–329.

[3] P. Berman and S. Hannenhalli, "Fast Sorting by Reversal", *Proceedings of 7th Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science (1996), Springer Verlag.

[4] P. Berman and M. Karpinski, "On Some Tighter Inapproximability Results", ECCC Report No. 29 (1998), University of Trier, 1998.

[5] M. Blanchette, personal communication.

[6] A. Caprara, "Sorting Permutations by Reversals and Eulerian Cycle Decompositions", *SIAM Journal on Discrete Mathematics* 12 (1999) 91–110.

[7] A. Caprara, "On the Tightness of the Alternating-Cycle Lower Bound for Sorting by Reversals", *Journal of Combinatorial Optimization* 3 (1999) 149–182.

[8] A. Caprara, G. Lancia and S.K. Ng, "A Column-Generation Based Branch-and-Bound Algorithm for Sorting by Reversals", in M. Farach-Colton, Fred S. Roberts, M. Vingron and M. Waterman (eds.) *Mathematical Support for Molecular Biology; DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 47 (1999) 213–226, AMS Press.

[9] G. Carpaneto, S. Martello and P. Toth, "Algorithms and Codes for the Assignment Problem", *Annals of Operations Research* 13 (1988) 193–223.

[10] D.A. Christie, "A 3/2 Approximation Algorithm for Sorting by Reversals", *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms* (1998) 244–252, ACM Press.

[11] U. Derigs, "Solving Non-Bipartite Matching Problems via Shortest Path Techniques", *Annals of Operations Research* 13 (1988) 225–261.

(i,j)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	27	17	18	26	19	8	27	0	0	3	16	19	23	23	27	0	0	17	2
2	27	0	26	26	26	26	25	6	27	27	27	27	26	25	26	26	27	27	26	27
3	17	26	0	25	25	4	18	25	17	17	18	24	7	26	24	28	17	17	0	18
4	18	26	25	0	27	26	17	24	18	18	19	4	25	16	20	26	18	18	25	17
5	26	26	25	27	0	25	27	27	26	26	26	28	26	27	26	18	26	26	25	26
6	19	26	4	26	25	0	21	26	19	19	20	26	7	25	24	27	19	19	4	20
7	8	25	18	17	27	21	0	25	8	8	8	18	19	21	23	27	8	8	18	7
8	27	6	25	24	27	26	25	0	27	27	26	25	27	23	23	28	27	27	25	27
9	0	27	17	18	26	19	8	27	0	0	3	16	19	23	23	27	0	0	17	2
10	0	27	17	18	26	19	8	27	0	0	3	16	19	23	23	27	0	0	17	2
11	3	27	18	19	26	20	8	26	3	3	0	17	20	23	22	27	3	3	18	5
12	16	27	24	4	28	26	18	25	16	16	17	0	24	16	21	26	16	16	24	16
13	19	26	7	25	26	7	19	27	19	19	20	24	0	27	26	28	19	19	7	20
14	23	25	26	16	27	25	21	23	23	23	23	16	27	0	17	25	23	23	26	23
15	23	26	24	20	26	24	23	23	23	23	22	21	26	17	0	27	23	23	24	23
16	27	26	28	26	18	27	27	28	27	27	27	26	28	25	27	0	27	27	28	27
17	0	27	17	18	26	19	8	27	0	0	3	16	19	23	23	27	0	0	17	2
18	0	27	17	18	26	19	8	27	0	0	3	16	19	23	23	27	0	0	17	2
19	17	26	0	25	25	4	18	25	17	17	18	24	7	26	24	28	17	17	0	18
20	2	27	18	17	26	20	7	27	2	2	5	16	20	23	23	27	2	2	18	0

1	Homo Sapiens
2	Albinaria Coerulea
3	Arbacia Lixula
4	Artemia Franciscana
5	Ascaris Suum
6	Asterina Pectinifera
7	Balanoglossus Carnosus
9	Cepaea Nemoralis
10	Cyprinus Carpio
11	Didelphis Virginiana
12	Drosophila Yakuba
13	Florometra Serratissima
14	Katharina Tunicata
15	Lumbricus Terrestris
16	Onchocerca Volvulus
17	Polypterus Ornatipinnis
18	Protopterus Dolloi
19	Strongylocentrotus Purpuratus
20	Struthio Camelus

Table 7: Reversal distances between mitochondrial genomes.

[12] M. Farach-Colton, Fred S. Roberts, M. Vingron and M. Waterman (eds.) *Mathematical Support for Molecular Biology; DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 47 (1999), AMS Press.

[13] A.A. Farley, "A Note on Bounding a Class of Linear Programming Problems, Including Cutting Stock Problems", *Operations Research* 38 (1990) 922–923.

[14] N. Franklin, "Conservation of genome form but not sequence in the transcription antitermination determinants of bacteriophages λ, $\phi 21$ and $P22$", *Journal of Molecular Evolution* 181 (1985) 75–84.

[15] H.N. Gabow, "An Efficient Implementation of Edmond's Algorithm for Maximum Matchings on Graphs", *Journal of the ACM* 23 (1975) 221–234.

[16] A.M.H. Gerards, "Matching", in M.O. Ball, T.L. Magnanti, C.L. Monma and G.L. Nemhauser (eds.), *Networks; Handbooks in Operations Research and Management Sciences*, North Holland (1995).

[17] M. Grötschel, L. Lovász and A. Schrijver, "The Ellipsoid Method and its Consequences in Combinatorial Optimization", *Combinatorica* 1 (1981), 169–197.

[18] S. Hannenhalli, personal communication.

[19] S. Hannenhalli and P.A. Pevzner, "Transforming Cabbage into Turnip (Polynomial Algorithm for Sorting Signed Permutations by Reversals)", *Journal of the ACM* 48 (1999) 1–27.

[20] S. Hannenhalli and P.A. Pevzner, "To Cut ... or Not to Cut (Applications of Comparative Physical Maps in Molecular Evolution)", *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms* (1996) 304–313, ACM Press.

[21] R.W. Irving and D.A. Christie, "Sorting by Reversals: a Conjecture of Kececioglu and Sankoff", Working Paper (1996), Dept. of Computer Science, University of Glasgow.

[22] H. Kaplan, R. Shamir and R.E. Tarjan, "Faster and Simpler Algorithm for Sorting Signed Permutations by Reversals", *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms* (1997), ACM Press.

[23] L.G. Khachian, "A Polynomial Algorithm for Linear Programming", *Soviet Mathematics Doklady* 20 (1979) 191–194.

[24] J. Kececioglu and D. Sankoff, "Efficient Bounds for Oriented Chromosome Inversion Distance", *Proceedings of 5th Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science 807 (1994) 307–325, Springer Verlag.

[25] J. Kececioglu and D. Sankoff, "Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement", *Algorithmica* 13 (1995) 180–210.

[26] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys, "Sequencing and Scheduling: Algorithms and Complexity", in S.C. Graves, P.H. Zipkin and A.H.G. Rinnooy Kan (eds.) *Logistics of Production and Inventory; Handbooks in Operations Research and Management Sciences*, North Holland (1993).

[27] A. Merhotra and M.A. Trick, "A Column Generation Approach for Exact Graph Coloring", *INFORMS Journal on Computing* 8 (1996) 344–354.

[28] M.W. Padberg and G. Rinaldi, "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems", *SIAM Review* 33 (1991) 60–100.

[29] P.A. Pevzner and M.S. Waterman, "Open Combinatorial Problems in Computational Molecular Biology", *Proceedings of the 3rd Israel Symposium on the Theory of Computing and Systems* (1995) 158–172, IEEE Computer Society Press.

[30] D. Sankoff, "Analytical approaches to genomic evolution", *Biochimie* 75 (1993) 409–413.

[31] D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B.F. Lang and R. Cedergren, "Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome", *Proc. Natl. Acad. Sci. USA* 89 (1992) 6575–6579.

[32] D. Sankoff, G. Sundaram and J. Kececioglu, "Steiner Points in the Space of Genome Rearrangements", *International Journal of Foundations of Computer Science* 7 (1996) 1–9.

[33] N. Tran, "An Easy Case of Sorting by Reversals", *Proceedings of 8th Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science 1264 (1997) 83–89, Springer Verlag.

[34] P.H. Vance, C. Barnhart, E.L. Johnson and G.L. Nemhauser, "Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound", *Computational Optimization and Applications* 3 (1994) 111–130.

[35] G.A. Watterson, W.J. Ewens, T.E. Hall and A. Morgan, "The Chromosome Inversion Problem", *Journal of Theoretical Biology* 99 (1982) 1–7.