Fast Practical Solution of Sorting by Reversals

Alberto Caprara* Giuseppe Lancia† See Kiong Ng‡

Abstract

We deal with the practical solution of the problem of *Sorting a permutation By Reversals* (SBR), which has relevant applications in computational biology. We present a successful approach based on the use of *Linear Programming* (LP). In particular, we deal with an LP relaxation with exponentially many variables, that can be handled by generating variables "on the fly", according to a so-called *column generation* scheme. A major advantage with respect to previous analogous approaches is that the subproblem to face in the column generation phase requires no longer the solution of min-cost general matching problems, but the solution of min-cost *bipartite* matching problems. Experiments show that, at least for our purposes, there is a speed-up of one order of magnitude 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 prove the worst-case ratio between the lower bound value obtained by our new method and previous ones.

The LP relaxation is used within a polynomial-time heuristic and an enumerative exact algorithm for SBR. We show 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 2-3 seconds, as well as provide a solution within 2% of the optimum for random instances with $n = 500$ within 10 minutes.

These results show that, although the problem is hard and the exact algorithm we propose has apparently exponential running time even on average (in particular, random instances with $n = 400$ seem to take months, if not much longer), the instances of practical interest can be solved to proven optimality very fast.

Key words: sorting by reversals, alternating-cycle decomposition, column generation, matching, experimental results.

1 Introduction

In this paper 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 $\rho_1,\dots,\rho_{d(\pi)}$ such that $\pi\rho_1\dots\rho_{d(\pi)} = \iota$. The optimal solution value $d(\pi)$ is called the *reversal distance* of π.

The main application of the problem is in computational biology. Let the order of the genes in two single-chromosome organisms be represented by two permutations π and τ of $\{1,\dots,n\}$. An inversion of the segment comprising the genes from the i-th to the j-th is represented by a reversal of the interval (i,j). A shortest sequence of reversals needed to transform π into τ is clearly equal to an optimal solution of SBR on $\tau^{-1}\pi$. Therefore, the solution of SBR yields a possible scenario to explain how an organism evolved from another, under the simplifying assumptions that inversions were the only rearrangement to occur, and that evolution required the minimum number of rearrangements. Even if these assumptions lead to some approximation, both are well-motivated. Indeed, on the one hand inversions are by far the most frequent type of rearrangement, and on the other rearrangements are very rare events.

With the more general aim of reconstructing an evolutionary tree (see e.g. [24]), 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 [23, 22], Bafna and Pevzner [1], Hannenhalli and Pevzner [17, 18], Caprara, Lancia and Ng [8], Berman and Hannenhalli [3], Irving and Christie [19], Tran [25], Kaplan, Shamir and Tarjan [20], 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 Max SNP-hard in [4]. References [23, 1, 10] present, respectively, polynomial-time 2-approximation, $\frac{7}{4}$-approximation, and $\frac{3}{2}$-approximation algorithms.

In this paper we are concerned about the practical solution of SBR. We present a successful approach based on the use of *Linear Programming* (LP). In particular,

*DEIS, Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy, e-mail: acaprara@deis.unibo.it

†DEI, Università di Padova, Via Gradenigo 6/A, 35131 Padova, Italy, e-mail: lancia@dei.unipd.it

‡Smithkline Beecham Pharmaceuticals R&D, Bioinformatics, New Frontiers Science Park (North), Third Avenue, Harlow, Essex CM19 5AW, UK, e-mail: skng@cs.cmu.edu

we deal with an LP relaxation with exponentially many variables, that can be handled by generating variables "on the fly", according to a so-called *column generation* scheme. A major advantage with respect to previous analogous approaches (see [23], [8]) is that the subproblem to face in the column generation phase requires no longer the solution of min-cost general matching problems, but the solution of min-cost *bipartite* matching problems. Experiments show that, at least for our purposes, there is a speed-up of one order of magnitude 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 prove the worst-case ratio between the lower bound value obtained by our new method and previous ones.

The LP relaxation is used within a polynomial-time heuristic and an enumerative exact algorithm for SBR. We show 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) instances within few milliseconds on a workstation. Moreover, we can solve to optimality random instances with $n = 100$ within 2-3 seconds, as well as provide a solution within 2% of the optimum for random instances with $n = 500$ within 10 minutes.

These results show that, although the problem is hard and the exact algorithm we propose has apparently exponential running time even on average (in particular, random instances with $n = 400$ seem to take months, if not much longer), the instances of practical interest can be solved to proven optimality very fast.

The paper is organized as follows. In Section 2 we survey known results in the literature, describing a combinatorial relaxation of SBR and the associated (natural) *Integer Linear Programming* (ILP) formulation, whose LP relaxation can be solved using general matching for column generation. Section 3 explains our new LP relaxation, showing that the associated column generation problem can be solved by bipartite matching and that, even in the worst case, the lower bound value obtained is at least 3/4 times that of the original LP relaxation. We also illustrate exact and heuristic algorithms based on the use of this LP relaxation. Finally, in Section 4 we present computational results both on real-world and random instances.

2 SBR and Alternating-Cycle Decompositions

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, $|B| = |Y| (= b(\pi))$. Figure 1 depicts the breakpoint graph associated with the permutation (4 2 1 3).

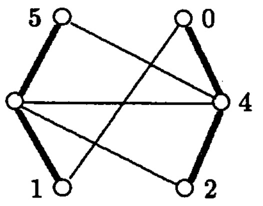

Figure 1: The breakpoint graph $G(\pi)$ associated with $\pi = (4\ 2\ 1\ 3)$. Gray edges are drawn as thin lines, black edges as thick lines.

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 $i = 1, \dots, m$;

(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 cycle 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. For a given π, let $c(\pi)$ be the maximum cardinality of an alternating-cycle decomposition of $G(\pi)$. Bafna and Pevzner [1] (see also Kececioglu and Sankoff [23]) proved the following property

THEOREM 2.1. *([1], [23]) For every permutation π, $b(\pi) - c(\pi) \leq d(\pi)$.*

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 following *Alternating-Cycle Decomposition* (ACD) problem, which, given the breakpoint graph $G(\pi)$ of a permutation π, calls for a maximum-cardinality alternating-cycle decomposition of $G(\pi)$.

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 was a strong enough motivation to encourage the use of this bound within a branch-and-bound framework. In principle, according to worst-case analysis, the lower bound can be quite far from the optimum, namely the following holds

THEOREM 2.2. *([6], [10]) For every permutation π, $b(\pi)-c(\pi) \geq \frac{2}{3}d(\pi)$. Moreover, there exist permutations for which $b(\pi) - c(\pi) = \frac{2}{3}d(\pi)$.*

In fact, a theoretical result which motivates the strength of the lower bound is

THEOREM 2.3. *([7]) The probability that $b(\pi) - c(\pi) < d(\pi)$ 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 [23, 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

$$\text{(2.1)} \qquad \max \sum_{C \in \mathcal{C}} x_C$$

subject to

$$\text{(2.2)} \qquad \sum_{C \ni e} x_C \leq 1, \qquad e \in E,$$

$$\text{(2.3)} \qquad x_C \in \{0,1\}, \qquad C \in \mathcal{C}.$$

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 (2.1)-(2.3), obtained by replacing constraints (2.3) with

$$\text{(2.4)} \qquad x_C \geq 0, \qquad C \in \mathcal{C}.$$

Solving the LP relaxation (2.1), (2.2) and (2.4) 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. This can be done by column generation techniques (see e.g. [2]), starting from a "small" LP with a restricted subset of the variables (implicitly fixing the other variables to 0), and iteratively solving the small LP, testing if the solution is optimal for the overall LP, and, if not, adding few variables with positive (in case of maximization) reduced cost to the small LP. The optimality test, namely the detection of variables with positive reduced cost, is called ***column generation problem.***

Due to a fundamental result of Grötschel, Lovàsz and Schrijver [15] based on Khachian's ellipsoid algorithm [21], the overall approach works in polynomial time provided the column generation problem can be solved efficiently. In particular, for LP (2.1), (2.2) and (2.4), it is easy to show that this problem, given a weight u_e associated with each edge $e \in E$, calls for finding an alternating cycle $C \in \mathcal{C}$ such that

$$\text{(2.5)} \qquad \sum_{e \in C} u_e^* < 1,$$

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

THEOREM 2.4. *([8]) LP (2.1), (2.2) and (2.4) can be solved in polynomial time (in n).*

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.

3 A Much Faster to Solve LP Relaxation

The solution of min-cost general matching problems in the solution of LP (2.1), (2.2) and (2.4) turns out to eat almost all of the overall computing time, even by using state-of-the-art codes for matching [14] and by trying a number of heuristic procedures to identify positive reduced-cost variables before resorting to the use of these codes. To overcome this serious drawback, we came up with an alternative LP relaxation, which is described in this section.

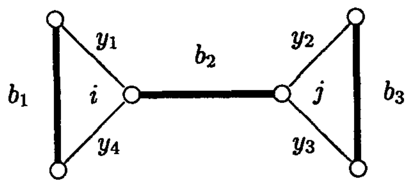

Figure 2: Pseudo alternating cycle $C = b_1, y_1, b_2, y_2, b_3, y_3, b_2, y_4$, where $\mu_{b_2,C} = 2$ and $\mu_{e,C} = 1$ for $e \in C \setminus \{b_2\}$.

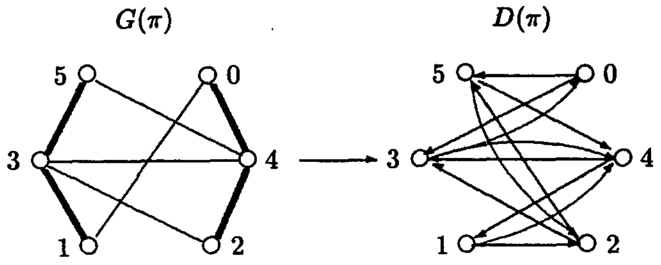

Figure 3: Graph $G(\pi)$ and the associated graph $D(\pi)$.

3.1 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 $b_1, y_1, b_2, y_2, \ldots, b_m, y_m$, such that:

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

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

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

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 $1 \le i < j \le m$. An example of a pseudo alternating cycles is given in Figure 2. 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 that we propose is the counterpart of (2.1), (2.2) and (2.4), 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

$$\max \sum_{C \in \mathcal{S}} x_C \tag{3.6}$$

subject to

$$\sum_{C \ni e} \mu_{e,C}\, x_C \le 1, \qquad e \in E \tag{3.7}$$

$$x_C \ge 0, \qquad C \in \mathcal{C}, \tag{3.8}$$

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. Note that the new LP relaxation is a relaxation of LP (2.1), (2.2) and (2.4) as the former has a wider set of variables. Obviously, $\mu_{e,C} = 1$ for each alternating cycle C and edge $e \in C$, and, by definition, $\mu_{e,C} \ge 2$ for at least one edge e in a pseudo alternating cycle C (in fact it is easy to show that $\mu_{e,C} \le 2$; see below). This yields the following

REMARK 3.1. *In every integer solution of (3.6)-(3.8), $x_C = 0$ for all $C \in \mathcal{P}$.*

Therefore, by solving LP (3.6)-(3.8) instead of (2.1), (2.2) and (2.4), 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, (3.6)-(3.8) yields an alternative ILP formulation of ACD.

3.2 Column generation by bipartite matching The dual of (3.6)-(3.8) has the form

$$\min \sum_{e \in E} u_e \tag{3.9}$$

subject to

$$\sum_{e \in C} \mu_{e,C}\, u_e \ge 1, \qquad C \in \mathcal{S}, \tag{3.10}$$

$$u_e \ge 0, \qquad e \in E. \tag{3.11}$$

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 $i, j \in V$, $D(\pi)$ 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 $u^*_{(i,k)}$. To be precise, there may exist two such k, in which case we consider the one leading to the minimum weight of arc (i,j). Figure 3 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 3.1. *Each surrogate alternating cycle of $G(\pi)$ corresponds to a dicycle of $D(\pi)$ of the same weight, and viceversa.*

Proof. Omitted.

COROLLARY 3.1. *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 $i \in V$, and initialize their weight to 0. The solution of an *Assignment Problem* (AP) on the weight matrix of $D(\pi)$ corresponds to a set of dicycles of $D(\pi)$ (possibly including loops) such that each node is visited by exactly one dicycle in the set and the sum of the weights of the arcs in the dicycles 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. [14]).

The solution of the AP corresponds to all the loop arcs, and has value 0. Note that, provided an ACD solution is contained in the set of variables in the current LP, the "interesting" alternating cycles visit at least one node of degree 4, as possible 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 loop arcs. Hence, we get the min-weight 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 [14] for details). Trying all the nodes of degree 4 in $G(\pi)$ guarantees finding, if any, a dicycle of $D(\pi)$ (and an alternating cycle of $G(\pi)$) of weight < 1, solving the column generation problem.

By using efficient implementations of the Hungarian method (see e.g. [14]), 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 3.2. *The column generation problem for LP (3.6)-(3.8) can be solved in $O(n^2 \log n)$ time.*

COROLLARY 3.2. *LP (3.6)-(3.8) can be solved in polynomial time (in n).*

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 (2.1), (2.2) and (2.4).

3.3 Worst-case comparison of the old and new lower bounds We now turn our attention to 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 (2.1), (2.2) and (2.4), and $\tilde{c}(\pi)$ the optimal solution value of LP (3.6)-(3.8). Clearly, $c^*(\pi) \le \tilde{c}(\pi)$, i.e. $\lceil b(\pi) - c^*(\pi) \rceil$ is a better lower bound on $d(\pi)$ than $\lceil b(\pi) - \tilde{c}(\pi) \rceil$. It is possible to show that 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 actual lower bounds on SBR is bounded by $\frac{3}{4}$, giving a partial theoretical explanation of empirical evidence.

LEMMA 3.1. *For every pseudo alternating cycle C of $G(\pi)$,*

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

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. Omitted.

THEOREM 3.3. *For every permutation π,*

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

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

Proof. Let $\tilde{c}(\pi) = \tilde{c}_{\mathcal{C}} + \tilde{c}_{\mathcal{P}}$, where $\tilde{c}_{\mathcal{C}}$ is the contribution to objective function (3.6) 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

$$\text{(3.12)} \qquad \tilde{c}(\pi) = \tilde{c}_{\mathcal{C}} + \tilde{c}_{\mathcal{P}} \ge c^*(\pi) \ge \tilde{c}_{\mathcal{C}}.$$

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 3.1 every pseudo alternating cycle of $G(\pi)$ contains at least 4 black edges. Adding up constraints (3.7) for black edges 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 imples 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 (3.6)-(3.8), say $\tilde{x}$,

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

Using the last inequality in (3.12), one gets

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

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

The illustration of the (asymptotically) tight examples is omitted.

3.4 Exact and heuristic algorithms based on the LP relaxation An effective heuristic algorithm for ACD is the following *diving* heuristic in which one solves LP (3.6)-(3.8), 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. Corollary 3.2 shows that this algorithm runs in polynomial time, as the number of variables to fix to 1 before termination is $O(n)$.

We next briefly outline an exact enumerative algorithm for ACD. *Branch-and-Price* (B&P) algorithms are branch-and-bound algorithms in which an LP relaxation with exponentially many variables is solved at every node of the branch-decision tree by column generation, see [2]. A B&P algorithm for ACD follows quite naturally from formulation (3.6)-(3.8). 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. For the specific case of ACD, such a branching rule amounts to replacing a node of degree 4 in $G(\pi)$ by two nodes of degree 2, each incident with a black and a grey edge, in the two possible ways.

A main point with the algorithms above is how to turn them into algorithms for SBR, which is the problem that we would like to solve. The main step in this direction is the very nice interpretation of an ACD solution in terms of *Signed SBR*, which is a relevant variant of SBR where elements have signs and the effect of a reversals is also to flip signs in the reversed subsequence. 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. A breakthrough result obtained by Hannenhalli and Pevzner [17] 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 [20], leading to the following

THEOREM 3.4. *([17], [3], [20]) SSBR can be solved in $O(n^2)$ time.*

We omit the description of the relationship between ACD solutions and SSBR and their use within our algorithms.

4 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.

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. [14]) 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 (2.1), (2.2) and (2.4) is the one by Cook and Rohe [11], which is the state-of-the-art for the solution of the problem.

Most of the following subsections will present results for random permutations. By *random permutation* with n elements we mean a permutation chosen with uniform probability among the $n!$ permutations with n elements.

4.1 Improvements with the new LP relaxation Table 1 reports the results obtained by solving instances associated with random permutations with our exact algorithm and with a variant of it that uses LP relaxation (2.1), (2.2) and (2.4), solving min-cost (nonbipartite) perfect matching problems in the column generation phase as illustrated in [8].

We solved instances up to size 200. 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 a time limit of 1 hour (column *# sol*), the solution time (*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 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.

Table 1 shows the clear improvement in running time (almost one order of magnitude for $n \geq 100$) achieved by using the new LP relaxation, whose lower bound value is essentially the same as that of the old LP relaxation.

4.2 Randomly generated instances Table 2 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 number of instances solved to proven optimality within a time limit of 1 hour (column *# sol*), 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 (*LP time*), the average time (average percentage over total time) for the solution of assignment problems in the column generation phase (*AP time*), 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 best algorithm [8], that could solve instances with size up to 100, which are now solved within 2-3 seconds. Within our time limit of 1 hour, we can consistently solve to proven optimality instances with n up to 200. To our knowledge, real-world instances are going to have a much smaller n (see also the next subsection), hence we expect our algorithm to be able to solve these instances to proven optimality within small computing time.

We tackled larger instances by applying our diving heuristic algorithm. The results are given in Table 3, where the columns have the same meaning as in Table 2, with the exception of column *final gap*, that reports the average (maximum) percentage gap between the solution found and the LP lower bound. Table 3 shows that even for instances with $n = 500$ we can find solutions provably within 2% of the optimum in about 10 minutes. Even if the size of the sample is very small, it seems that the running time of our heuristic algorithm in practice is basically $\Theta(n^4)$.

4.3 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 given us by Sridhar Hannenhalli [16]. The input format is a partially signed permutation, meaning that the orientation of only part of the genes (namely, 47 out of 138) within the genomes is known. From this input we derived two instances, one by ignoring the signs of the elements and the other by considering the actual partially signed permutation. Table 4 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 2. 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 5, 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 to compare a pair. This shows that, even if the problem id NP-hard, 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. [24]).

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 4.3. 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] W.J. Cook and A. Rohe, "Computing Minimum-Weight Perfect Matchings", *INFORMS Journal on Computing* 11 (1999) 138–148.

[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] 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.

[14] 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).

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

[16] S. Hannenhalli, personal communication.

[17] S. Hannenhalli and P.A. Pevzner, "Transforming Cabbage into Turnip (Polynomial Algorithm for Sorting Signed Permutations by Reversals)", *Proceedings of the 27th Annual ACM Symposium on the Theory of Computing* (1995) 178–187, ACM Press.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

[24] 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.

[25] 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.

n	old LP				new LP				$\frac{b(\pi)-\bar{c}(\pi)}{b(\pi)-c^*(\pi)}$	speed-up
	# sol	time (match time)	# nodes	root time	# sol	time (AP time)	# nodes	root time		
25	10	0.01 (0.00)	1.0	0.01	10	0.01 (0.00)	1.1	0.01	1.000	1.7
50	10	0.16 (0.11)	1.0	0.12	10	0.07 (0.03)	1.0	0.04	0.999	2.6
75	10	2.58 (2.19)	3.0	0.84	10	0.52 (0.15)	2.8	0.25	1.000	3.3
100	10	21.65 (19.86)	13.2	3.88	10	2.78 (0.95)	13.9	0.55	1.000	7.1
125	10	133.60 (125.08)	42.5	9.08	10	4.83 (1.77)	9.1	1.26	1.000	7.2
150	10	251.96 (239.48)	30.8	13.76	10	22.50 (9.53)	32.8	2.23	1.000	6.2
175	6	1919.67 (1783.37)	320.4	24.41	10	400.93 (195.24)	506.7	3.33	1.000	7.3
200	4	2687.93 (2552.16)	188.8	34.84	9	956.55 (479.55)	826.0	5.18	1.000	6.7

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

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

n	# singl	# rev	time	LP time	AP time	final gap
100	95.1 (100)	76.5 (79)	1.10 (1.50)	0.47 (43.2%)	0.31 (28.6%)	0.8% (2.5%)
200	196.8 (200)	159.5 (160)	16.85 (19.42)	4.74 (28.1%)	7.21 (42.8%)	0.9% (1.9%)
300	295.6 (300)	245.3 (247)	88.52 (93.75)	20.28 (22.9%)	39.51 (44.6%)	1.6% (2.4%)
400	395.0 (400)	330.2 (333)	270.43 (287.80)	60.73 (22.5%)	123.18 (45.6%)	1.6% (2.1%)
500	496.0 (500)	417.5 (422)	649.30 (667.13)	143.32 (22.1%)	301.85 (46.5%)	1.9% (2.6%)

Table 3: Heuristic solution of random permutations. Average (maximum) 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 4: Results for man and mouse genome instances.

(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 5: Mitochondrial genome distances. All permutations have size 37 and the overall time for computing the table was 4.5 seconds.