A COLUMN–GENERATION BASED BRANCH–AND–BOUND ALGORITHM FOR SORTING BY REVERSALS

Alberto Caprara*, Giuseppe Lancia+ and See–Kiong Ng°

* DEIS, University of Bologna, Italy

+ GSIA, Carnegie–Mellon University, USA

° Computer Science Department, Carnegie–Mellon University, USA

1 Introduction

Until recently, most evolutionary studies in molecular biology have been based on sequence alignment, i.e. comparison of single genes to detect local mutations in the sequence of nucleotides. However, in the last few years, we have witnessed an increasing interest in analyzing entire genomes at once, this way shifting the attention from gene level to chromosome level (Sankoff et al., [13], Sankoff, [12]). In fact, as it is often found that the order of genes is preserved more easily than the DNA sequence (see [3], [14]), looking at genomes instead of DNA sequences allows one building some otherwise extremely difficult evolutionary scenarios; this is particularly true for plant mitochondrial DNA, virology and Drosophila genetics. Rearrangement of genomes can occur in many different ways, among which inversions, transpositions, deletions, insertions and duplications of fragments.

Let the order of the genes in two single–chromosome organisms be given by permutations $\pi = (\pi_1 \ \dots \ \pi_n)$ and $\tau = (\tau_1 \ \dots \ \tau_n)$ of $\{1, \dots, n\}$. An inversion of the segment comprising the genes from the i–th to the j–th is represented by the permutation $\rho = (1 \ \dots \ i-1 \ \ j \ \dots \ i \ \ j+1 \ \dots \ n)$. We call ρ a *reversal* of the interval (i, j). Composition of π with ρ yelds $\pi\rho = (\pi_1 \ \dots \ \pi_{i-1} \ \pi_j \ \dots \ \pi_i \ \pi_{j+1} \ \dots \ \pi_n)$ where genes $\pi_i, \dots, \pi_j$ have been reversed. The *reversal distance* between π and τ is the minimum integer k such that there exist reversals $\rho_1, \dots, \rho_k$ for which $\pi\rho_1 \dots \rho_k = \tau$.

Clearly the reversal distance between π and τ equals the reversal distance between $\tau^{-1}\pi$ and the identity permutation ι. *Sorting a permutation π by reversals* (SBR) is the problem of finding the reversal distance $d(\pi)$ between π and ι (and a sequence of reversals $\rho_1, \dots, \rho_{d(\pi)}$ for which $\pi\rho_1 \dots \rho_{d(\pi)} = \iota$).

The algorithmic study of the problem of sorting by reversals has been started by Kececioglu and Sankoff [11] who gave the first exact and approximation algorithms and also posed a set of challenging open questions, first of all the computational complexity of the problem. Later Bafna and Pevzner [1] proved some of their conjectures and developed an approximate algorithm which yields solutions never worse than 7/4 times the optimum.

Although the major question remains to determine whether sorting by reversals is NP–hard, recently Hannenhalli and Pevzner [7] surprisingly found that a very similar problem in which we consider *signed* permutations is polynomially solvable, and described an $O(n^4)$ algorithm for solving it. A signed permutation $\widehat{\pi}$ is a permutation in which each element is signed either +

or $-$. In this case a reversal has the effect of both inverting the order of a sequence of elements and also flipping their signs. This model is more realistic when applied to evolutionary studies since genes do have an orientation, although this is not always known. The problem of sorting by reversals a signed permutation $\widehat{\pi}$ amounts to finding a shortest sequence of reversals that transforms $\widehat{\pi}$ into the signed identity permutation $(+1 + 2 \ \ldots \ + n)$.

In studying the problem of signed permutations, Hannenhalli and Pevzner were also able to identify the bottleneck making the unsigned version hard in the presence of elements π_i for which $|\pi_i - \pi_{i-1}| \neq 1$ and $|\pi_i - \pi_{i+1}| \neq 1$, called *singletons* of π. They then proved that SBR is polynomially solvable when π contains $O(\log n)$ singletons, and showed that sorting by reversals is in fact equivalent to finding proper signs for such elements, reducing the problem to a signed one.

In this paper we present an exact branch–and–bound algorithm for SBR. Our lower bound is based on the results of Bafna and Pevzner [1] (see also Kececioglu and Sankoff [11]) where the reversal distance is related to the number of edge–disjoint cycles in a suitable graph associated with the permutation. In order to estimate this number we solve a *Linear Programming* (LP) problem containing a possibly exponential (in n) number of variables. This LP is solved by using *column generation* techniques, which have been shown to be effective for many combinatorial optimization problems (see [2]). The approach is described in Sections 2.1 and 2.2, where we prove that the above–mentioned LP is solvable in polynomial time. The branching is based on signing the singletons of π, as discussed in detail in Section 2.3. Finally, in order to obtain upper bounds (feasible solutions) we use a greedy heuristic, also described in section 2.3.

The overall procedure turns out to be very effective; the tightness of the lower bound obtained by using column generation has allowed us to solve problems of considerable size if compared to the previous algorithms.

The only exact algorithm so far proposed for SBR and widely computationally tested is due to Kececioglu and Sankoff [11]. The algorithm was implemented before Hannenhalli and Pevzner [8] proved some properties which could be used for improving its performance. Kececioglu and Sankoff report in their paper optimal solutions for problems of up to $n = 50$ elements. With our algorithm we have been able to solve exactly problems on random permutations with 100 elements, in a matter of a few minutes. Tables with detailed computational results are reported in Section 3. A promising result at the very beginning of our work was being able to solve in a few minutes (now in a few seconds) the unsolved 36–element problem mentioned in [11].

2 The Branch–and–Bound Algorithm

In this section we give a description of the algorithm we use for solving SBR. Subsections 2.1, 2.2 and 2.3 outline the fundamental parts of the algorithm, while Subsection 2.4 gives some details concerning its actual implementation.

2.1 Lower Bound

The lower bound we compute for SBR is based on a result by Bafna and Pevzner [1], which is also very closely related to the work of Kececioglu and Sankoff [11].

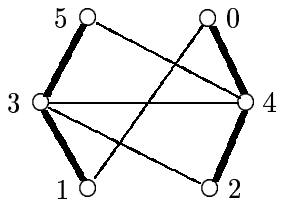

Figure 1: The breakpoint graph $G(\pi)$ associated with $\pi = (4\ 2\ 1\ 3)$.

Consider the permutation $\pi = (\pi_1\ \ldots\ \pi_n)$, and ideally add to π the elements $\pi_0 := 0$ and $\pi_{n+1} := n+1$, re–defining $\pi := (0\ \pi_1\ \ldots\ \pi_n\ n+1)$. Let the *inverse permutation* π^{-1} of π be defined by $\pi^{-1}_{\pi_i} := i$ for $i = 0, \ldots, n+1$. Following the description in [1], we define the *breakpoint graph* $G(\pi) = (V, E \cup F)$ as follows. Each node $v \in V$ represents an element of π, therefore we let $V := \{0, \ldots, n+1\}$. E is the set of *red* (say) edges, each of the form (π_i, π_{i+1}), for all $i \in \{0, \ldots, n\}$ 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 $\sigma := (0\ 1\ \ldots\ n\ n+1)$. Such a pair (π_i, π_{i+1}) is called a *breakpoint* of π. F is the set of *blue* (say) edges, each of the form $(i, i+1)$, for all $i \in \{0, \ldots, n\}$ such that $|\pi^{-1}_i - \pi^{-1}_{i+1}| \neq 1$, i.e. elements which are in consecutive positions in σ but not in π. Notice that each node $i \in V$ has either degree 0, 2 or 4, and has the same number of blue and red edges incident with it. We let $\delta(i) \subseteq E \cup F$ denote the set of edges incident with node i. Figure 2.1 depicts the breakpoint graph associated with permutation (4 2 1 3).

An *alternating cycle* of $G(\pi)$ is a sequence of edges $C = \{e_1, f_1, e_2, f_2, \ldots, e_m, f_m\}$, where $e_i \in E$, $f_i \in F$ for $i = 1, \ldots, m$; e_i and f_j have a common node for $i = j = 1, \ldots, m$ and for $i = j+1$, $j = 1, \ldots, m$ (where $e_{m+1} := e_1$); and $e_i \neq e_j$, $f_i \neq f_j$ for $1 \leq i < j \leq m$. A node $i \in V$ such that $|\delta(i) \cap C| > 0$ is called *visited* by C. 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 2.1. A *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 a cycle decomposition. A cycle decomposition of maximum cardinality is called *optimal*. In the graph of Figure 2.1, cycles $\{(0,4),(4,3),(3,1),(1,0)\}$ and $\{(4,2),(2,3),(3,5),(5,4)\}$ form an optimal cycle decomposition. Furthermore, Bafna and Pevzner [1] (see also Kececioglu and Sankoff [11]) proved the following property.

Theorem 1 *([1], [11]) Let $d(\pi)$ be the optimal solution value to SBR for a given π, and let $c(\pi)$ be the cardinality of an optimal cycle decomposition of $G(\pi)$. Then $d(\pi) \geq |E| - c(\pi)$.*

Therefore $|E| - c(\pi)$ gives a valid lower bound on the optimal solution value to SBR. In practical cases this bound turns out to be very tight, and is frequently equal to the optimum, as observed by Kececioglu and Sankoff [11].

Unfortunatley, we do not know how to compute $c(\pi)$ in polynomial time. Instead, we compute an upper bound $c^*(\pi)$ on $c(\pi)$, and use $|E| - c^*(\pi)$ as a valid lower bound on $d(\pi)$. For this purpose, let us first state the maximum–cardinality cycle decomposition problem as the following *Integer Linear Programming* (ILP) problem. Let $\mathcal{C}$ denote the set of all the alternating cycles of $G(\pi)$, and for each $C \in \mathcal{C}$ introduce a 0–1 decision variable x_C. A possible

ILP model is

$$c(\pi) := \max \sum_{C \in \mathcal{C}} x_C \tag{1}$$

s.t.

$$\sum_{C \ni e} x_C = 1 \quad e \in E \tag{2}$$

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

Constraints (2) ensure that each edge in E is contained in exactly one cycle C such that $x_C = 1$. It is easy to see that the same will hold for each edge in F, due to the structure of $G(\pi)$. In an optimal solution x^* to (1)–(3), the set of cycles C such that $x^*_C = 1$ forms a maximum–cardinality cycle decomposition.

A valid lower bound $c^*(\pi)$ on $c(\pi)$ can be obtained by solving the *LP relaxation* of model (1)–(3), obtained by substituting constraints (3) with

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

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

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

s.t.

$$\sum_{e \in C} y_e \geq 1 \quad C \in \mathcal{C} \tag{6}$$

where each variable y_e, $e \in E$, is unconstrained in sign.

Solving the LP relaxation (1), (2) and (4) amounts to solving an LP problem having $|E| = O(n)$ constraints, and $|\mathcal{C}| = O(2^n)$ variables, i.e. a possibly huge number of variables. Kececioglu and Sankoff [11] face the same problem with the LP relaxation for the maximum–cardinality cycle decomposition of a slightly different graph than $G(\pi)$. They solve an LP containing only variables associated with cycles up to a fixed length, and use a suitably–defined correction term so as to obtain an upper bound on the solution value for the LP containing all the variables.

In fact, due to a result by Grötschel, Lovàsz and Schrijver [5] that uses Khachian's ellipsoid algorithm [9], LP (1), (2) and (4) can be solved in polynomial time (in n) provided the following *column generation problem* is efficiently solvable: given a weight y^*_e for each $e \in E$, find an alternating cycle $C \in \mathcal{C}$ such that

$$\sum_{e \in C} y^*_e < 1, \tag{7}$$

or prove that none exists. In the next section we describe a polynomial–time algorithm for this problem, thus proving the following statement.

Proposition 1 *LP (1), (2) and (4) can be solved in polynomial time.*

We call an alternating cycle satisfying (7) a *violated cycle* (w.r.t. edge weights y^*). Solving the column generation problem amounts to finding (if any) a negative reduced–cost variable x_C, with respect to the dual variables y^*_e. The overall LP relaxation can then be solved through the following scheme:

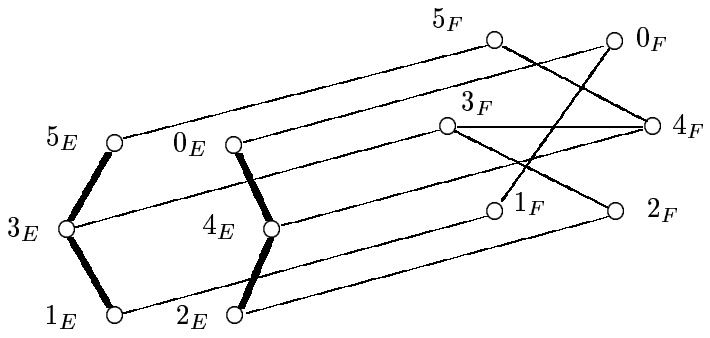

Figure 2: The graph $H(G(\pi))$ associated with $\pi = (4\ 2\ 1\ 3)$.

procedure SOLVE_LP_RELAX;
begin
initialize the LP (1), (2) and (4) by considering only a subset $\mathcal{S} \subset \mathcal{C}$ of the variables (e.g., a heuristically–determined cycle decomposition);
repeat
solve the LP (1), (2) and (4) with variables in $\mathcal{S}$ only and let y^* be an optimal dual solution;
for all $e \in E$ let y^*_e be the weight of edge e;
if violated cycles exist, find a nonempty subset $\mathcal{V}$ of them and let $\mathcal{S} := \mathcal{S} \cup \mathcal{V}$
until no violated cycle exists
end.

In practice, solving LP relaxations having an exponential number of variables by column generation turns out to be a very effective way of approaching many difficult combinatorial optimization problems, see the survey by Barnhart et al. [2].

2.2 Column Generation

The identification of a violated cycle (with respect to y^*) in $G(\pi)$ can be stated as the problem of finding an alternating cycle of $G(\pi)$ having weight < 1, where each edge $e \in E$ is given the weight y^*_e, and each edge $f \in F$ is given the weight 0. Notice that edge weights are possibly negative. We next show how to solve this problem in polynomial time.

Let us call an alternating cycle C *simple* if $|\delta(i) \cap C| = 0$ or 2 for all $i \in V$. Our reduction starts with a general result, which does not apply only to breakpoint graphs, but to all graphs with edges of two colors. Let $G = (V, E \cup F)$ be such a graph.

Consider the following construction, analogous to that used by Grötschel and Pulleyblank for finding minimum–weight odd and even paths in an undirected graph [6]. Define the graph $H(G) = (V_E \cup V_F, L_E \cup L_F \cup L_V)$ from G as follows. For each node $i \in V$, $H(G)$ contains 2 *twin* nodes i_E, i_F. For each red edge $(i, j) \in E$, $H(G)$ has an edge $(i_E, j_E) \in L_E$; for each blue edge $(i, j) \in F$, $H(G)$ has an edge $(i_F, j_F) \in L_F$. Each edge in $L_E \cup L_V$ is given the same weight as its counterpart in G. Finally twin nodes are connected in $H(G)$ by means of edges $(i_E, i_F) \in L_V$, for each $i \in V$, each having weight 0. For example, the graph $H(G(\pi))$ associated with $G(\pi)$ in Figure 2.1, is depicted in Figure 2.2.

A *matching* M of $H(G)$ is a subset of edges of $H(G)$ such that no node of $H(G)$ is incident with more than one edge in M. The following proposition justifies the use of $H(G)$.

Proposition 2 *There is a bijection between perfect matchings of $H(G)$ and (possibly empty) sets of node–disjoint simple alternating cycles of G.*

Proof. To map matchings into cycles, consider a perfect matching M in $H(G)$. If $M = L_V$, i.e. M is formed by all the edges of the form (i_E, i_F), there is no corresponding cycle in G. Otherwise, consider a node i_E which is not matched with i_F. The matching then contains a sequence of edges of the form (i_E, j_E),(j_F, k_F),(k_E, l_E),...,(p_F, i_F), such that the corresponding edges of G form a simple alternating cycle. If we are given a different perfect matching $M' \neq L_V$, then at least one edge in $M' \cap (L_E \cup L_F)$ is not in M; therefore the alternating cycles corresponding to M and M' are different. To show the map is onto, consider a set of simple alternating cycles. For each node $i \in V$ which is not spanned by this set, match i_E with i_F, and match the remaining nodes of $H(G)$ by using the edges in $L_E \cup L_F$ corresponding to the edges in the cycles. This way a perfect matching is obtained. □

We have seen that sets of simple cycles in G and perfect matchings in H are equivalent; however we want to allow also for alternating cycles which go through some of the nodes twice. Since we are interested in graphs $G(\pi)$ where each node has degree ≤ 4, for this purpose we can define a new graph G_2 from G by applying the following two steps (fig. 2.2):

i) *separation*: for each pair i, j of nodes of degree 4 in G connected by an edge, we introduce two new nodes, say k, l, between them, and replace the edge (i, j) by the equivalent alternating path $(i, k), (k, l), (l, j)$, where (i, k) and (l, j) have the same color as (i, j). Let G_1 be the resulting graph. Note that there is an obvious bijection between the alternating cycles of G and G_1. Also, no two nodes of degree 4 are adjacent in G_1.

ii) *splitting*: for each node of degree 4, we perform the following *split* operation. Assume the neighbors of i are j, k (through blue edges) and l, m (through red edges); then we introduce a new node i' in V and blue edges (i', j), (i', k). Finally we remove the edge (i, m) and replace it by a red edge (i', m). Let us call the graph thus obtained G_2.

Note that for each red edge e of G_2, either $e = (i, m)$ and e is in G_1 also, or $e = (i', m)$ and (i, m) is in G_1. From the way splitting is defined, we have a bijection ϕ between red edges of G_1 and G_2. Furthermore, it can be easily seen that two red edges e', e'' are linked via a blue edge f in G_1 (i.e. e', f, e'' is an alternating path of G_1) if and only if their correspondents are linked via a blue edge in G_2. As a final remark, note that all alternating cycles in G_2 are simple as in G_2 each node has degree ≤ 3.

Proposition 3 *There is a bijection between alternating cycles (simple or not) of G and G_2.*

Proof. By our previuos remarks $e_1, e_2, \ldots, e_k$ are the consecutive red edges of an alternating cycle in G_2 if and only if $\phi(e_1), \phi(e_2), \ldots, \phi(e_k)$ are the consecutive red edges of an alternating (though not necessarily simple) cycle of G_1. Since two different alternating cycles have at least one different red edge, we get that there is a bijection of alternating cycles of G_2 and G_1. Since there is also a bijection between alternating cycles of G_1 and G, the claim follows. □

In view of propositions 2 and 3, we can state the following

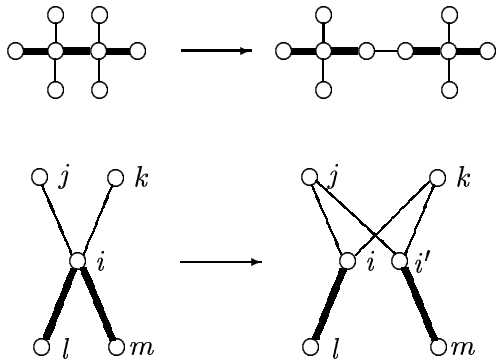

Figure 3: Separation and Splitting.

Proposition 4 *There is a bijection between perfect matchings of $H(G_2)$ and (possibly empty) sets of edge–disjoint alternating cycles of G.*

Again, note that to node–disjoint cycles of G_2 there correspond edge–disjoint cycles of G_1 (and G). Indeed, in G_2 there are at most two copies of any (blue) edge of G_1, which are incident with a common node and therefore cannot be both present in two node–disjoint cycles of G_2.

Having established a correspondence between matchings of $H(G_2)$ and sets of alternating cycles in G, we now turn to considering edge weights. The way we update the weights in passing from G to G_2 is as follows:

i) *separation*: if the edge e removed is blue, we give weight 0 to all the edges in the new 3–path; if it is red, we give weight $y_e{}^*$ to one of the red edges in the new 3–path (arbitrarily chosen) and weight 0 to the other two new edges.

ii) *splitting*: referring to the previously described situation, we give weight 0 to the blue edges (i', j), (i', k) and weight $y^*_{(i,m)}$ to the red edge (i', m).

It is immediate to see that for any nonempty set $\mathcal{S}$ of edge–disjoint alternating cycles of G and the corresponding matching M of $H(G_2)$ we have $w(M) = y^*(\mathcal{S})$, where $w(M) := \sum_{e \in M} w(e)$ and $y^*(\mathcal{S})$ is the overall weight of the cycles in $\mathcal{S}$. It then follows:

Proposition 5 *There is one alternating cycle in G of weight < 1 if and only if there exists a perfect matching $M \neq L_V$ in $H(G_2)$ of weight < 1.*

To find such a matching we can proceed as follows. First, we compute a *Minimum Weight Perfect Matching* (MWPM) in $H(G_2)$. If it does not consist only of edges in L_V, then clearly it has negative weight and we are done. Otherwise, for each egde $e \in L_E$ of weight $w(e) < 1/2$ (there must be at least one in a cycle of weight < 1) we find a MWPM on $H(G_2)$ which includes edge e. This can be accomplished by removing all other edges in $H(G_2)$ incident with e. We stop as soon as a matching of weight < 1 is found, or otherwise we conclude that no violated cycle exists.

The size of G_2 is greater than the size of G by at most a constant factor. Also, a MPWM can be solved in polynomial time, and our procedure requires computing at most $O(n)$ MWPMs. Then Proposition 1 follows when we substitute $G(\pi)$ for G.

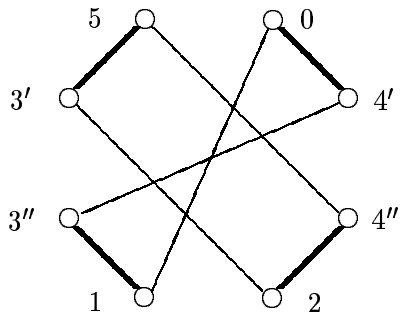

Figure 4: The breakpoint graph $G^S(\pi^S)$ associated with $\pi^S = (+4\ 2\ 1\ -3)$.

2.3 Branching and Heuristics

Our branching and heuristic procedures were inspired by the work on the *signed* version of SBR by Kececioglu and Sankoff [10], and further motivated by the recent relevant results of Hannenhalli and Pevzner [7], [8].

Given π, call π_i a *singleton* if $|\pi_i - \pi_{i-1}| \neq 1$ and $|\pi_i - \pi_{i+1}| \neq 1$. Notice that singletons correspond to nodes of degree 4 in $G(\pi)$. The *signing* of a singleton π_i corresponds to imposing a parity on the number $\rho(\pi_i)$ of reversals involving element π_i in a solution to SBR. In particular, we say that π_i is signed $+$ if $\rho(\pi_i)$ has to be even, and that π_i is signed $-$ if $\rho(\pi_i)$ has to be odd. Hannenhalli and Pevzner [7], [8] proved the following result.

Theorem 2 *([7], [8]) SBR with signs imposed on each singleton can be solved in polynomial time.*

Based on this result, Hannenhalli and Pevzner suggest to solve SBR by finding the optimal signing to singletons. For this purpose, they propose a simple enumerative algorithm. Our branch–and–bound implements the same idea in a more effective way.

We follow again the construction in [1]. Given a permutation π^S of n elements where some singletons are signed, we construct the associated breakpoint graph $G^S(\pi^S) = (V^S, E^S \cup F^S)$ from $G(\pi)$, where π is the unsigned counterpart of π^S. In $G(\pi)$, each singleton π_i has incident red edges (π_i, π_{i-1}) and (π_i, π_{i+1}), and incident blue edges $(\pi_i, \pi_i - 1)$ and $(\pi_i, \pi_i + 1)$. Initialize $G^S(\pi^S) := G(\pi)$. For each signed singleton π_i of π^S, perform the following *splitting* operation on node π_i of $G^S(\pi^S)$. In V^S replace node π_i by nodes π_i' and π_i''. In F^S replace edge $(\pi_i, \pi_i - 1)$ by $(\pi_i', \pi_i - 1)$ and $(\pi_i, \pi_i + 1)$ by $(\pi_i'', \pi_i + 1)$. Moreover, if π_i is signed $+$, in E^S replace edge (π_i, π_{i-1}) by (π_i', π_{i-1}) and (π_i, π_{i+1}) by (π_i'', π_{i+1}). Similarly, if π_i is signed $-$, in E^S replace edge (π_i, π_{i-1}) by (π_i'', π_{i-1}) and (π_i, π_{i+1}) by (π_i', π_{i+1}). Figure 2.3 depicts the breakpoint graph $G^S(\pi^S)$, for $\pi^S := (+4\ 2\ 1\ -3)$.

From the results of Bafna and Pevzner [1], Hannenhalli and Pevzner [7], [8], it is immediate to see that $|E^S| - c^S(\pi^S)$ is a valid lower bound on the optimal solution value to the signed version of SBR, where $c^S(\pi^S)$ is the maximum cardinality of a cycle decomposition of $G^S(\pi^S)$. Furthermore, if all singletons are signed in π^S, every node of $G^S(\pi^S)$ has degree 0 or 2, and then there exist a unique (optimal) cycle decomposition, which is trivially determined.

We observe that an optimal cycle decomposition of $G(\pi)$ gives somehow an indication on the optimal signing of each singleton, as the following proposition suggests.

Proposition 6 *Given a permutation π and a corresponding optimal cycle decomposition of $G(\pi)$ of cardinality $c(\pi)$, it is easy to sign all the singletons so that $c^S(\pi) = c(\pi)$ for the resulting signed permutation π^S.*

Proof. Construct the signed permutation π^S from π as follows. Consider each singleton π_i of π, in turn. If the optimal cycle decomposition for $G(\pi)$ contains only one alternating cycle C visiting π_i, sign π as you like. Otherwise, let C be the cycle visiting π_i that contains edge (π_i, π_{i-1}). If C contains edge $(\pi_i, \pi_i - 1)$, sign π_i +, otherwise sign it −. It easy to check that each cycle in the optimal decomposition for $G(\pi)$ has a counterpart in $G^S(\pi^S)$. □

Therefore, once an optimal cycle decomposition for $G(\pi)$ is available, it is possible to construct a signed permutation π^S from π, such that the lower bounds on the optimal solution values to signed SBR for π^S and to (unsigned) SBR for π coincide. Hopefully, the signed and unsigned problems have a same sequence of reversals as an optimal solution.

Our heuristic algorithm works as follows: given the (possibly fractional) solution x^* to the LP relaxation (1), (2) and (4), we greedily sign the singletons by taking into account this solution, and find a greedy heuristic solution to the corresponding signed problem. Each singleton π_i is signed by considering the cycle C visiting π_i having highest value x^*_C. If C contains all the 4 edges incident with π_i, then any sign is given to π_i. Otherwise, π_i is signed so that C has a counterpart in the graph associated with the signed permutation (see the proof of Proposition 6).

The branching rule is based on the same idea. If the LP solution is integer (i.e. the cycle decomposition problem has been solved to optimality), we choose a singleton visited by 2 cycles in the solution (if any), and generate two subproblems by signing it + and −, respectively. For one subproblem, the lower bound will be unchanged, according to Proposition 6. For the other subproblem, one can hopefully have a lower bound increase. If no singleton is visited by 2 cycles in the solution, we branch by signing any singleton + and −, respectively, with no bound increase for both subproblems generated.

If the LP solution is fractional (i.e. the cycle decomposition problem has not been solved yet), we branch by trying to "break" the fractional solution so as to get quickly to an integer one, hopefully with a lower bound increase on both branches. We use the following simple property.

Proposition 7 *Any alternating cycle C such that $0 < x^*_C < 1$ in the LP solution, visits at least one singleton.*

We then choose the variable x_C having value closest to 0.5, consider a singleton π_i in cycle C, and branch by signing π_i + and −, respectively.

Therefore, in the branch decision tree, at the root node we have no signed singleton, at the intermediate nodes we have a subset of the singletons which are signed, and at the leaf nodes we have all singletons signed, so that the corresponding problem can be solved efficiently (without further branching) by the algorithm of Hannenhalli and Pevzner [7], [8].

2.4 Implementation Details

Our algorithm was coded in ANSI C, and ran on a SUN SparcStation 300. We use CPLEX 2.1 as LP solver, which is known as a very fast and robust LP solver, capable of dealing with the degeneracy problems sometimes arising for the largest instances we solved.

In the column generation phase, we apply a slightly different procedure than that described in Section 2.2. This procedure is much faster in practice, but possibly returns "degenerate" violated cycles. More precisely, we construct the arc–weighted directed graph $D(\pi) = (V, A)$ from $G(\pi)$ and from a dual solution y^* to (5)–(6), as follows. $D(\pi)$ has the same node set V as $G(\pi)$. 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 $(i, k) \in E$ and $(k, j) \in F$. The arc weight for (i, j) is given by $y^*_{(i,k)}$. (In other words, each path consisting of a red and a blue edge in $G(\pi)$ is replaced by an arc in $D(\pi)$.) It is easy to see that if $D(\pi)$ does not contain any cycle having weight < 1, then no violated cycle exists in $G(\pi)$. On the other hand, not all the cycles of $D(\pi)$ correspond to alternating cycles of $G(\pi)$, since they possibly correspond to sequences of edges where some edges appear twice. If all the cycles of weight < 1 identified in $D(\pi)$ have this "degenerate" form, the corresponding variables are added to the LP (1), (2) and (4), which is therefore relaxed. Nevertheless, in practical cases we typically obtain the same lower bound values as if we used the exact column generation procedure of Section 2.2.

We briefly describe how we check the existence of cycles 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 cycles in $D(\pi)$. If none is found, we consider each node 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. This gives a minimum–weight cycle in $G(\pi)$ visiting node i. This new AP need not be solved from scratch, since by starting from the solution of the first AP (with all loop weights $= 0$), the computation of an augmenting path is sufficient. The AP's are solved by using the code by Carpaneto, Martello and Toth [4], the complexity of the overall procedure being $O(n^3)$.

Our tree exploration in the branch–and–bound algorithm is of *best first* type, i.e. among the active subproblems we explore first the "sons" of the subproblem having the lowest lower bound value. At the root node, we solve LP (1), (2) and (4) starting with a set of variables given by a heuristically–determined cycle decomposition. Each other subproblem inherits all the LP variables of its "parent", apart from those corresponding to cycles eliminated by the branching constraints.

At the leaf nodes, the signed version of SBR is solved by signing the elements which are not singletons according to the rules illustrated in [8], and by optimally sorting the (fully–signed) resulting permutation by using the branch–and–bound algorithm of Kececioglu and Sankoff [10]. In fact, in our experiments we never reach leaf nodes of the tree.

The heuristic algorithm is applied at every tree node, and sorts a signed permutation with a sequence of reversals, each randomly chosen among those removing the maximum number of breakpoints (see [10] for the definition of breakpoint of a signed permutation, and for a detailed description of the signed problem). Notice that our final aim is sorting an unsigned permutation, therefore we avoid reversals acting on a single element.

n	singl	B&B nodes	root gap	B&B time	root time
10	6.46	1.16	0.06	0.23 s	0.22 s
20	16.28	1.08	0.06	1.03 s	0.98 s
30	25.68	1.40	0.20	3.67 s	3.27 s
40	35.82	1.58	0.52	10.79 s	9.32 s
50	45.96	2.58	0.84	27.58 s	20.81 s
60	55.70	2.46	1.18	49.57 s	39.77 s
70	66.26	5.28	1.34	1 m 51.57 s	1 m 12.18 s
80	75.86	7.58	1.98	3 m 23.04 s	1 m 59.97 s
90	86.02	10.38	2.48	5 m 52.91 s	3 m 9.32 s
100	95.96	11.66	3.30	9 m 3.33 s	4 m 38.88 s

Table 1: Computational results for randomly–generated instances (average values over 50 instances – SUN SparcStation 300 CPU times).

3 Computational Results

As mentioned in the introduction, the algorithm described in the previous section was initially tested, among others, on the real–world 36–element permutation mentioned in [11]. In that paper, the authors observed that their exact algorithm was not capable of solving the instance, yielding an upper bound of 27 and a lower bound of 25 on the optimal solution value. In fact, in [8] an optimal solution of value 26 is given (without reporting the computing time).

At the very beginning of our implementation, we observed that optimally solving the LP relaxation (1), (2) and (4), yields a lower bound value of 25, corresponding to a widely fractional LP solution. After a few branchings, the best lower bound value for the active subproblems is 26. As soon as we introduced our heuristic algorithm, we were able to find the optimal solution and proving optimality in a few minutes on a SUN SparcStation 300. With our current implementation we can solve the problem in a few seconds.

In Table 3 we report computational results for randomly–generated SBR instances, corresponding to permutations where for $i = 1, \ldots, n$ the element π_i is chosen with uniform probability among $\{1, \ldots, n\} \setminus \{\pi_1, \ldots, \pi_{i-1}\}$. The entries of the table have the following meaning:

n is the number of elements in the permutations;

singl is the average number of singletons;

B&B nodes is the average number of subproblems explored by the branch–and–bound algorithm;

root gap is the average gap between the upper and lower bound values at the root node;

B&B time is the average time needed by the branch–and–bound algorithm;

root time is the average time spent at the root node of the branch–and–bound algorithm.

The average values are over 50 instances for each value of n run on a SUN SparcStation 300. All the instances were solved to optimality by our algorithm. The computational results compare favourably with those reported in [11], where not all instances were solved to optimality for $n \geq 40$, in particular no instance was solved for $n \geq 60$. Notice that most of the computing

time is spent at the root node: it is basically required for solving the initial LP, with a lot of calls to the column generation procedure. (At the other nodes, the solution of the LP is considerably faster, since a set of "good" LP variables is available from the previous nodes.)

References

[1] V. Bafna and P. Pevzner, "Genome Rearrangements and Sorting by Reversals", *Proceedings of the 34th IEEE Symposium on Foundations of Computer Science* (1993) 148–157 (to appear on *SIAM Journal on Computing*).

[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", in *Mathematical Programming: State of the Art 1994* (1994) 186–207, edited by J.R. Birge and K.G. Murty.

[3] 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.

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

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

[6] M. Grötschel and W.R. Pulleyblank, "Weakly Bipartite Graphs and the Max–Cut Problem", *Operations Research Letters* 1 (1981) 23–27.

[7] S. Hannenhalli and P. 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), to appear.

[8] S. Hannenhalli and P. Pevzner, "Reversals Do Not Cut Long Strips", Technical Report CSE-95-006, Department of Compter Science and Engineering, The Pennsylvania State University, February 1995.

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

[10] 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, Spinger Verlag.

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

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

[13] 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.

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