A FASTER AND SIMPLER ALGORITHM FOR SORTING SIGNED PERMUTATIONS BY REVERSALS *

HAIM KAPLAN†, RON SHAMIR‡, AND ROBERT E. TARJAN§

Abstract. We give a quadratic-time algorithm for finding the minimum number of reversals needed to sort a signed permutation. Our algorithm is faster than the previous algorithm of Hannenhalli and Pevzner and its faster implementation of Berman and Hannenhalli. The algorithm is conceptually simple and does not require special data structures. Our study also considerably simplifies the combinatorial structures used by the analysis.

AMS (MOS) subject classification: 62P10 68P10

Key words: sorting permutations, reversal distance, computational molecular biology.

1. Introduction. In this paper we study the problem of sorting signed permutations by reversals. A *signed permutation* is a permutation $\pi = (\pi_1, \ldots, \pi_n)$ on the integers $\{1, \ldots, n\}$, where each number is also assigned a sign of plus or minus. A *reversal*, $\rho(i, j)$, on π transforms π to

$$\pi' = \pi\rho(i, j) = $$
$$(\pi_1, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n).$$

The minimum number of reversals needed to transform one permutation to another is called the *reversal distance* between them. The problem of *sorting signed permutations by reversals* is to find, for a given signed permutation π, a sequence of reversals of minimum length that transforms π to the identity permutation $(+1, +2, \ldots, +n)$.

The motivation to studying the problem arises in molecular biology: Concurrent with the fast progress of the Human Genome Project, genetic and DNA data on many model organisms is accumulating rapidly, and consequently the ability to compare genomes of different species has grown dramatically. One of the best ways of checking similarity between genomes on a large scale is to compare the order of appearance of identical genes in the two species. In the Thirties, Dobzhansky and Sturtevant [7] had already studied the notion of inversions in chromosomes of *drosophila*. In the late Eighties, Jeffrey Palmer demonstrated that different species may have essentially the same genes, but the gene orders may differ between species. Taking an abstract perspective, the genes along a chromosome can be thought of as points along a line. Numbers identify the particular genes; and, as genes have directionality, signs correspond to their direction. Palmer and others have shown that the difference in order may be explained by a small number of reversals [17, 18, 19, 20, 12]. These reversals correspond to evolutionary changes during the history of the two genomes, so the num-

*A preliminary version of this paper was presented at the Eighth ACM-SIAM Symposium on Discrete Algorithms [13].

†AT&T-labs research, 180 Park Ave, Florham Park, NJ 07932 USA. hkl@research.att.com

‡ Department of Computer Science, Sackler Faculty of Exact Sciences, Tel Aviv University, Tel-Aviv 69978 ISRAEL. Research supported in part by a grant from the Ministry of Science and the Arts, Israel, and by US Department of Energy, grant No. DE-FG03-94ER61913/A000. shamir@math.tau.ac.il

§Department of Computer Science, Princeton University, Princeton, NJ 08544 USA and InterTrust Technologies Corporation, Sunnyvale, CA 94086 USA. Research at Princeton University partially supported by the NSF, Grants CCR-8920505 and CCR-9626862, and the Office of Naval Research, Contract No. N00014-91-J-1463. ret@cs.princeton.edu

ber of reversals reflects the evolutionary distance between the species. Hence, given two such permutations, their reversal distance measures their evolutionary distance.

Mathematical analysis of genome rearrangement problems was initiated by Sankoff [22, 21]. Kececioglu and Sankoff [16] gave the first constant-factor polynomial approximation algorithm for the problem and conjectured that the problem is NP-hard. Bafna and Pevzner [3], and more recently Christie [6] improved the approximation factor, and additional studies have revealed the rich combinatorial structure of rearrangement problems [15, 14, 2, 9, 10]. Quite recently, Caprara [5] has established that sorting *unsigned* permutations is NP-hard, using some of the combinatorial tools developed by Bafna and Pevzner [3].

In 1995, Hannenhalli and Pevzner [11] showed that the problem of sorting a *signed* permutation by reversals is polynomial. They proved a duality theorem that equates the reversal distance with the sum of three combinatorial parameters (see Theorem 2.3 below). Based on this theorem, they proved that sorting signed permutations by reversals can be done in $O(n^4)$ time. More recently, Berman and Hannenhalli [4] described a faster implementation that finds a minimum sequence of reversals in $O(n^2\alpha(n))$ time, where α is the inverse of Ackerman's function [1] (see also [23]).

In this study we give an $O(n^2)$ algorithm for sorting a signed permutation of n elements, thereby improving upon the previous best known bound [4]. In fact, if the reversal distance is r, our algorithm requires $O(r \cdot n + n\alpha(n))$ time. In addition to giving a better time bound, our work considerably simplifies both the algorithm and combinatorial structure needed for the analysis, as follows:

- The basic object we work with is an implicit representation of the overlap graph, to be defined later, in contrast with the interleaving graph in [11] and [4]. The overlap graph is combinatorially simpler than the interleaving graph. As a result, it is easier to produce a representation for the overlap graph from the input, and to maintain it while searching for reversals.
- As a consequence of our ability to work with the overlap graph we need not perform any "padding transformations", nor do we have to work with "simple permutations" as in [11] and [4].
- We deal with the unoriented and oriented parts of the permutation separately, which makes the algorithm much simpler.
- The notion of a *hurdle*, one of the combinatorial entities defined by [11] for the duality theorem, is simplified and is handled in a more symmetric manner.
- The search for the next reversal is much simpler, and requires no special data structures. Our algorithm computes connected components only once, and any simple implementation of it suffices to obtain the quadratic time bound. In contrast, in [4] a logarithmic number of connected component computations may be performed per reversal, using the union-find data structure.

The paper is organized as follows: Section 2 gives the necessary preliminaries. Section 3 gives an overview of our algorithm. Sections 4 and 5 give the details of our algorithm. We summarize our results and suggest some further research in Section 6.

2. Preliminaries. This section gives the basic background, primarily the theory of Hannenhalli and Pevzner, on which we base our algorithm. The reader may find it helpful to refer to Figure 2.1, in which the main definitions are illustrated. We start with some definitions for unsigned permutations. Let $\pi = (\pi_1, \ldots, \pi_n)$ denote a permutation of $\{1, \ldots, n\}$. Augment π to a permutation on $n+2$ vertices by adding $\pi_0 = 0$ and $\pi_{n+1} = n+1$ to it. A pair (π_i, π_{i+1}), $0 \leq i \leq n$ is called a *gap*. Gaps are classified into two types. A gap (π_i, π_{i+1}) is a *breakpoint* of π if and only if

$|\pi_i - \pi_{i+1}| > 1$; otherwise, it is an *adjacency* of π. We denote by $b(\pi)$ the number of breakpoints in π.

A *reversal*, $\rho(i,j)$, on a permutation π transforms π to

$$\begin{aligned} &\pi' = \pi\rho(i,j) = \\ &(\pi_1, \ldots, \pi_{i-1}, -\pi_j, -\pi_{j-1}, \ldots, -\pi_i, \pi_{j+1}, \ldots, \pi_n). \end{aligned}$$

We say that a reversal $\rho(i,j)$ *acts on* the gaps (π_{i-1}, π_i) and (π_j, π_{j+1}).

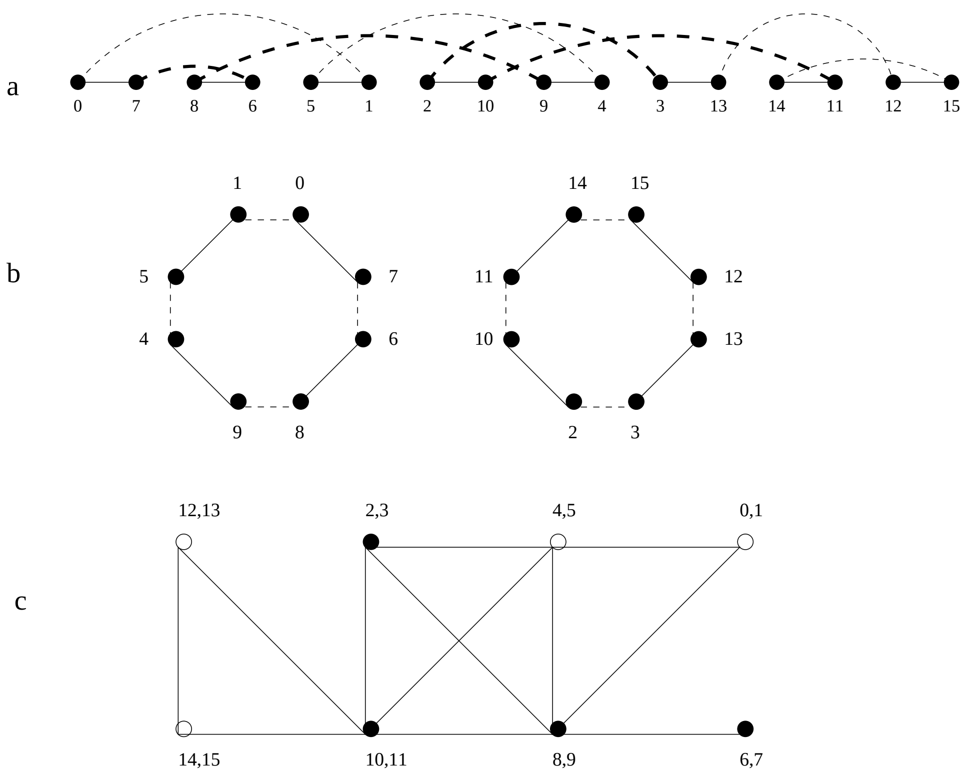

FIG. 2.1. *a) The breakpoint graph, $B(\pi)$, of the permutation $\pi = (4, -3, 1, -5, -2, 7, 6)$. Black edges are solid; gray edges are dashed; oriented edges are bold. b) $B(\pi)$ decomposes into two disjoint alternating cycles. c) The overlap graph, $OV(\pi)$. Black vertices correspond to oriented edges.*

2.1. The breakpoint graph. The *breakpoint graph* $B(\pi)$ of a permutation $\pi = (\pi_1, \ldots, \pi_n)$ is an edge-colored graph on $n+2$ vertices $\{\pi_0, \pi_1, \ldots, \pi_{n+1}\} = \{0, 1, \ldots, n+1\}$. We join vertices π_i and π_j by a *black edge* if (π_i, π_j) is a breakpoint in π and by a *gray edge* if (i,j) is a breakpoint in π^{-1}.

We define a one-to-one mapping u from the set of signed permutations of order n into the set of unsigned permutations of order $2n$ as follows. Let π be a signed permutation. To obtain $u(\pi)$, replace each positive element x in π by $2x-1, 2x$ and each negative element $-x$ by $2x, 2x-1$. For any signed permutation π, let $B(\pi) = B(u(\pi))$. Note that in $B(\pi)$ every vertex is either isolated or incident to exactly one black edge and one gray edge. Therefore, there is a unique decomposition of $B(\pi)$ into cycles. The edges of each cycle alternate between gray and black. Call a reversal $\rho(i,j)$ such that i is odd and j even an *even reversal*. The reversal $\rho(2i+1, 2j)$

on $u(\pi)$ mimics the reversal $\rho(i+1, j)$ on π. Thus, sorting π by reversals is equivalent to sorting the unsigned permutation $u(\pi)$ by even reversals. Henceforth we will consider the latter problem, and by a reversal we will always mean an even reversal. Let $b(\pi) = b(u(\pi))$ and let $c(\pi)$ be the number of cycles in $B(\pi)$.

Figure 2.1(a) shows the breakpoint graph of the permutation $\pi = (4, -3, 1, -5, -2, 7, 6)$. It has eight breakpoints and decomposes into two alternating cycles, i.e. $b(\pi) = 8$, and $c(\pi) = 2$. The two cycles are shown in Figure 2.1(b). Figure 2.2(a) shows the breakpoint graph of $\pi' = (4, -3, 1, 2, 5, 7, 6)$, which has seven breakpoints and decomposes into two cycles.

For an arbitrary reversal ρ on a permutation π, define $\Delta b(\pi, \rho) = b(\pi\rho) - b(\pi)$ and $\Delta c(\pi, \rho) = c(\pi\rho) - c(\pi)$. When the reversal ρ and the permutation π are clear from the context, we will abbreviate $\Delta b(\pi, \rho)$ by Δb and $\Delta c(\pi, \rho)$ by Δc. As Bafna and Pevzner [3] observed, the following values are taken by Δb and Δc depending on the types of the gaps $\rho(i, j)$ acts on:

1. Two adjacencies: $\Delta c = 1$ and $\Delta b = 2$.
2. A breakpoint and an adjacency: $\Delta c = 0$ and $\Delta b = 1$.
3. Two breakpoints each belonging to a different cycle: $\Delta b = 0$, $\Delta c = -1$.
4. Two breakpoints of the same cycle C:
   a. (π_i, π_{j+1}) and (π_{i-1}, π_j) are gray edges: $\Delta c = -1$, $\Delta b = -2$.
   b. Exactly one of (π_i, π_{j+1}) and (π_{i-1}, π_j) is a gray edge: $\Delta c = 0$, $\Delta b = -1$.
   c. Neither (π_i, π_{j+1}) nor (π_{i-1}, π_j) is a gray edge, and when breaking C at i and j vertices $i-1$ and $j+1$ end up in the same path: $\Delta b = 0$, $\Delta c = 0$.
   d. Neither (π_i, π_{j+1}) nor (π_{i-1}, π_j) is a gray edge, and when breaking C at i and j vertices $i-1$ and $j+1$ end up in different paths: $\Delta b = 0$, $\Delta c = 1$.

Call a reversal *proper* if $\Delta b - \Delta c = -1$, i.e. it is either of type 4a, 4b, or 4d. We say that a reversal ρ *acts on* a gray edge e if it acts on the breakpoints which correspond to the black edges incident with e. A gray edge is *oriented* if a reversal acting on it is proper, otherwise it is *unoriented*. Notice that a gray edge (π_k, π_l) is oriented if and only if $k + l$ is even. For example, the gray edge $(0, 1)$ in the graph of Figure 2.1(a) is unoriented, while the gray edge $(7, 6)$ is oriented.

2.2. The overlap graph. Two intervals on the real line *overlap* if their intersection is nonempty but neither properly contains the other. A graph G is an *interval overlap graph* if one can assign an interval to each vertex such that two vertices are adjacent if and only if the corresponding intervals overlap (see, e.g., [8]). For a permutation π, we associate with a gray edge (π_i, π_j) the interval $[i, j]$. The *overlap graph* of a permutation π, denoted $OV(\pi)$, is the interval overlap graph of the gray edges of $B(\pi)$. Namely, the vertex set of $OV(\pi)$ is the set of gray edges in $B(\pi)$, and two vertices are connected if the intervals associated with their gray edges overlap. We shall identify a vertex in $OV(\pi)$ with the edge it represents and with its interval in the representation. Thus, the endpoints of a gray edge are actually the endpoints of the interval representing the corresponding vertex in $OV(\pi)$. Note that all the endpoints of intervals in this representation are distinct integers. A connected component of $OV(\pi)$ that contains an oriented edge is called an *oriented component*; otherwise, it is called an *unoriented component*.

Figure 2.1(c) shows the interval overlap graph for $\pi = (4, -3, 1, -5, -2, 7, 6)$. It has only one oriented component. Figure 2.2(b) shows the overlap graph of the permutation $\pi' = (4, -3, 1, 2, 5, 7, 6)$, which has two connected components, one oriented and the other unoriented.

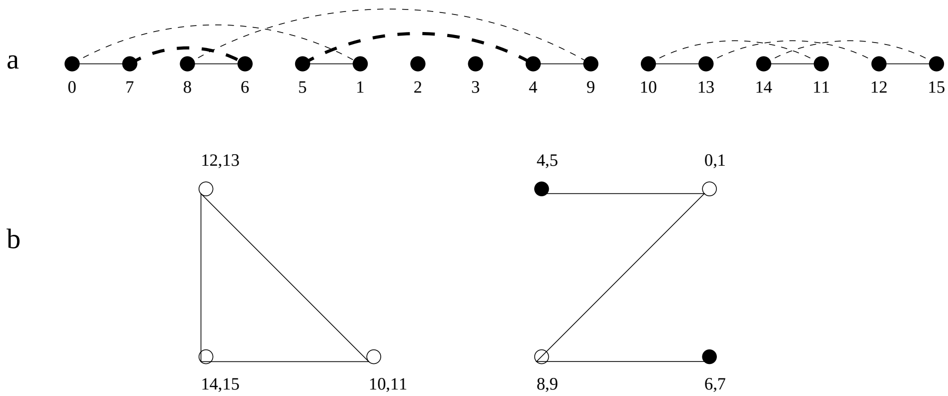

FIG. 2.2. *a) The breakpoint graph of $\pi' = (4, -3, 1, 2, 5, 7, 6)$. π' was obtained from π of Figure 2.1 by the reversal $\rho(7, 10)$; or, equivalently, by the reversal defined by the gray edge $(2, 3)$. b) The overlap graph of π'.*

2.3. The connected components of the overlap graph. Let X be a set of gray edges in $B(\pi)$. Define $\min(X) = \min\{i \mid (\pi_i, \pi_j) \in X\}$, $\max(X) = \max\{j \mid (\pi_i, \pi_j) \in X\}$ and $span(X) = [\min(X), \max(X)]$. Equivalently, one can look at the interval overlap representation of $OV(\pi)$ mentioned above and define the span of a set of vertices X as the minimum interval which contains all the intervals of vertices in X.

The major object our algorithm will work with is $OV(\pi)$, though for efficiency considerations we will avoid generating it explicitly. In contrast, Pevzner and Hannenhalli worked with the *interleaving graph* H_π, whose vertices are the alternating cycles of $B(\pi)$. Two cycles C_1 and C_2 are connected by an edge in H_π iff there exists a gray edge $e_1 \in C_1$ and a gray edge $e_2 \in C_2$ that overlap.

The following lemma and its corollary imply that the partition imposed by the connected components of $OV(\pi)$ on the set of gray edges is identical to the one imposed by the connected components of H_π:

LEMMA 2.1. *If M is a set of gray edges in $B(\pi)$ that corresponds to a connected component in $OV(\pi)$ then $\min(M)$ is even and $\max(M)$ is odd.*

Proof. Assume $\min(M)$ is odd. Then $\pi_{\min(M)} + 1$ and $\pi_{\min(M)} - 1$ must both be in $span(M)$ (i.e. there exist $l_1, l_2 \in span(M)$ such that $\pi_{l_1} = \pi_{\min(M)} + 1$ and $\pi_{l_2} = \pi_{\min(M)} - 1$). Thus $\pi_{\min(M)}$ is neither the maximum nor the minimum element in the set $\{\pi_i \mid i \in span(M)\}$. Hence, either the maximum element or the minimum element in $span(M)$ is π_j for some $\min(M) < j < \max(M)$. By the definition of $B(\pi)$ there must be a gray edge (π_j, π_l) for some $l \notin span(M)$, contradicting the fact that M is a connected component in $OV(\pi)$. The proof that $\max(M)$ is odd is similar. $\square$

As an illustration of Lemma 2.1, consider Figure 2.2(a). Let $M_1 = \{(0, 1), (4, 5), (8, 9), (6, 7)\}$ and $M_2 = \{(10, 11), (12, 13), (14, 15)\}$. Then $span(M_1) = [0, 9]$ and $span(M_2) = [10, 15]$.

COROLLARY 2.2. *Every connected component of $OV(\pi)$ corresponds to the set of gray edges of a union of cycles.*

Proof. Assume by contradiction that C is a cycle whose gray edges belong to at least two connected components in $OV(\pi)$. Assume M_1 and M_2 are two of these components such that there are two consecutive gray edges $e_1 \in M_1$ and $e_2 \in M_2$ along C. Since the spans of different connected components in $OV(\pi)$ cannot overlap there are two different cases to consider.

1. $span(M_2) \subseteq span(M_1)$ (the case $span(M_1) \subseteq span(M_2)$ is symmetric). Since e_1 and e_2 are in different components they cannot overlap. Thus, either the right endpoint of e_2 is even and equals $\max(M_2)$ or the left endpoint of e_2 is odd and equals $\min(M_2)$. In both cases we have a contradiction to Lemma 2.1.
2. $span(M_2)$ and $span(M_1)$ are disjoint intervals. W.l.o.g. assume that $\max(M_1) < \min(M_2)$. The right endpoint of e_1 is even and equals $\max(M_1)$, which contradicts Lemma 2.1. □

Note that in particular Corollary 2.2 implies that an overlap graph cannot contain isolated vertices.

2.4. Hurdles. Let $\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k}$ be the subsequence of $0, \pi_1, \ldots, \pi_n, n+1$ consisting of those elements incident with gray edges that occur in unoriented components of $OV(\pi)$. Order $\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k}$ on a circle CR such that π_{i_j} follows $\pi_{i_{j-1}}$ for $2 \leq j \leq k$ and π_{i_1} follows π_{i_k}. Let M be an unoriented connected component in $OV(\pi)$. Let $E(M) \subset \{\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_k}\}$ be the set of endpoints of the edges in M. An unoriented component M is a *hurdle* if the elements of $E(M)$ occur consecutively on CR.

This definition of a hurdle is different from the one given by Hannenhalli and Pevzner [11]. It is simpler in the sense that minimal hurdles and the maximal one do not have to be treated in different ways. Using Corollary 2.2 above, one can prove that the hurdles as we have defined them are identical to the ones defined by Hannenhalli and Pevzner. Let $h(\pi)$ denote the number of hurdles in a permutation π.

A hurdle is *simple* if when one deletes it from $OV(\pi)$ no other unoriented component becomes a hurdle, and it is a *super hurdle* otherwise. A *fortress* is a permutation with an odd number of hurdles all of which are super hurdles.

The following theorem was proved by Hannenhalli and Pevzner.

THEOREM 2.3. *[11] The minimum number of reversals required to sort a permutation π is $b(\pi) - c(\pi) + h(\pi)$, unless π is a fortress, in which case exactly one additional reversal is necessary and sufficient.*

3. Overview of our algorithm. Denote by $d(\pi)$ the reversal distance of π, i.e., $d(\pi) = b(\pi) - c(\pi) + h(\pi) + 1$ if π is a fortress and $d(\pi) = b(\pi) - c(\pi) + h(\pi)$ otherwise.

Following the theory developed in [11], it turns out that given a permutation π with $h(\pi) > 0$ one can perform $t = \lceil h(\pi)/2 \rceil$ reversals and transform π into a permutation π' such that $h(\pi') = 0$ and $d(\pi') = d(\pi) - t$. If $OV(\pi)$ has unoriented components then our algorithm first finds t such reversals that transform π into a π' which has only oriented components.

Our method of "clearing the hurdles" uses the theory developed by Hannenhalli and Pevzner. In Section 5 we describe an efficient implementation of this process which uses the implicit representation of the overlap graph $OV(\pi)$. Our implementation runs in $O(n)$ time assuming $OV(\pi)$ is already partitioned into its connected components. Recently, Berman and Hannenhalli [4] gave an $O(n\alpha(n))$ algorithm for computing the connected components of an interval overlap graph given implicitly by its representation. Using their algorithm we can clear the hurdles from a permutation in $O(n\alpha(n))$ time.

The overlap graph of π', $OV(\pi')$, has only oriented components. In Section 4 we prove that in the neighborhood of any oriented gray edge e there is an oriented gray edge e_1 (e_1 could be the same as e) such that a reversal acting on e_1 does not create new hurdles. Call such a reversal a *safe reversal*. We develop an efficient algorithm to locate a safe reversal in a permutation with at least one oriented gray edge. Our algorithm uses only an implicit representation of the overlap graph and runs in $O(n)$ time.

The second stage of our algorithm repeatedly finds a safe reversal and performs it as long as $OV(\pi)$ is not empty. Clearly the overall complexity is $O(r \cdot n + n\alpha(n))$, where r is the number of reversals required to sort π'.

3.1. Representing the overlap graph. We assume that the input is given as a sequence of n signed integers representing π^0. First the permutation $\pi = u(\pi^0)$ is constructed as described in Section 2.1 and stored in an array. We also construct an array representing π^{-1}. It is straightforward to verify that with these two arrays we can determine for each element in π whether it is a left or a right endpoint of a gray edge in constant time. In case the element is an endpoint of a gray edge we can also find the other endpoint and check whether the edge is oriented in constant time.

Thus the arrays π and π^{-1} comprise a representation of $OV(\pi)$. Our algorithm will maintain these two arrays while carrying out the reversals that it finds. The time to update the arrays is proportional to the length of the interval being reversed, which is $O(n)$. We shall give a high-level presentation of our algorithm and use primitives like "Scan the oriented gray edges in increasing left endpoint order". It is easy to see how to implement these primitives using the arrays π and π^{-1}; we shall omit the details.

It is easy to produce a list of the intervals in the representation of $OV(\pi)$ sorted by either left or right endpoint from the arrays π and π^{-1}. It is also possible to maintain them without increasing the asymptotic time bound of the algorithm. In practice it may be faster to maintain such lists instead of, or in addition to π and π^{-1}.

4. Eliminating oriented components. First we introduce some notation. Recall that the vertices of $OV(\pi)$ are the gray edges of $B(\pi)$. In order to avoid confusion we will usually refer to them as vertices of $OV(\pi)$. Hence a vertex of $OV(\pi)$ is *oriented* if the corresponding gray edge is oriented and it is *unoriented* otherwise. Let e be a vertex in $OV(\pi)$. Denote by $r(e)$ the reversal acting on the gray edge corresponding to e. Denote by $N(e)$ the set of neighbors of e in $OV(\pi)$ including e itself. Denote by $ON(e)$ the subset of $N(e)$ containing the oriented vertices and by $UN(e)$ the subset of $N(e)$ containing the unoriented vertices.

In this section we prove that if an oriented vertex e exists in $OV(\pi)$ then there exists an oriented vertex $f \in ON(e)$ such that $r(f)$ is proper and safe. We also describe an algorithm that finds a proper safe reversal in a permutation that contains at least one oriented edge.

We start with the following useful observation:

OBSERVATION 4.1. *Let e be a vertex in $OV(\pi)$ and let $\pi' = \pi r(e)$. $OV(\pi')$ could be obtained from $OV(\pi)$ by the following operations. 1) Complement the graph induced by $OV(\pi)$ on $N(e) - \{e\}$, and flip the orientation of every vertex in $N(e) - \{e\}$. 2) If e is oriented in $OV(\pi)$ then remove it from $OV(\pi)$. 3) If there exists an oriented edge e' in $OV(\pi)$ with $r(e) = r(e')$ then remove e' from $OV(\pi)$.*

Note that if e is an oriented vertex in a component M of $OV(\pi)$, $M - \{e\}$ may split into several components in $OV(\pi')$. (Compare figures 2.1(c) and 2.2(b).) Denote these components by $M'_1(e), \ldots, M'_k(e)$, where $k \geq 1$. We will refer to $M'_i(e)$ simply as M'_i whenever e is clear from the context.

Let C be a clique of oriented vertices in $OV(\pi)$. We say that C is *happy* if for every oriented vertex $e \notin C$ and every vertex $f \in C$ such that $(e, f) \in E(OV(\pi))$ there exists an oriented vertex $g \notin C$ such that $(g, e) \in E(OV(\pi))$ and $(g, f) \notin E(OV(\pi))$.

For example, in the overlap graph shown in Figure 2.1(c) $\{(2,3),(10,11)\}$ and $\{(6,7)\}$ are happy cliques, but $\{(2,3),(10,11),(8,9)\}$ is not. Our first theorem claims that one of vertices in any happy clique defines a safe proper reversal.

THEOREM 4.1. *Let C be a happy clique and let e be a vertex in C such that $|UN(e')| \leq |UN(e)|$ for every $e' \in C$. Then the reversal $r(e)$ is safe.*

Proof. Let $\pi' = \pi r(e)$ and assume by contradiction that $M'_i(e)$ is unoriented for some $1 \leq i \leq k$. Clearly $N(e) \cap M'_i \neq \emptyset$.

Assume there exists $y \in N(e) \cap M'_i$ such that $y \notin C$. Clearly y must be oriented in $OV(\pi)$ and since C is happy it must also have an oriented neighbor y' such that $(y', e) \notin E(OV(\pi))$. Since y' is not adjacent to e in $OV(\pi)$ it stays oriented and adjacent to y in $OV(\pi')$, in contradiction with the assumption that M'_i is unoriented. Hence we may assume that $N(e) \cap M'_i \subseteq C$.

Let $y \in N(e) \cap M'_i$ and let $z \in UN(e)$. Vertex z is oriented in $OV(\pi')$ and if it is adjacent to y in $OV(\pi')$ we obtain a contradiction. Hence, z and y are not adjacent in $OV(\pi')$, so they must be adjacent in $OV(\pi)$. Hence we obtain that $UN(e) \subseteq UN(y)$ in $OV(\pi)$. Corollary 2.2 implies that component M'_i cannot contain y alone. Thus y must have a neighbor x in M'_i. Since $N(e) \cap M'_i \subseteq C$, vertex x is not adjacent to e in $OV(\pi)$. Thus we obtain that $(x,y) \in OV(\pi)$, $(x,e) \notin OV(\pi)$, and x is unoriented in $OV(\pi)$. Since we have already proved that $UN(e) \subseteq UN(y)$, this implies that $UN(e) \subset UN(y)$, in contradiction with the choice of e. □

For example Theorem 4.1 implies that the reversal defined by the gray edge $(10,11)$ is a safe proper reversal for the permutation of Figure 2.1 (a), since it corresponds to the vertex with maximum unoriented degree in the happy clique $\{(2,3),(10,11)\}$. On the other hand, the reversal defined by $(2,3)$ creates a new unoriented component, as it yields the permutation shown in Figure 2.2.

The following theorem proves that a happy clique exists in the neighborhood of any oriented edge.

THEOREM 4.2. *Let e be an oriented vertex in $OV(\pi)$. There exists an oriented vertex $f \in ON(e)$ such that for $\pi' = \pi r(f)$, all the components in $OV(\pi')$ are oriented.*

Proof. By Theorem 4.1 it suffices to show that there exists a happy clique C in $ON(e)$.

Let $Ext(e) = \{x \in ON(e) \mid \text{ there exists } y \in ON(x) \text{ such that } y \notin ON(e)\}$. That is, $Ext(e)$ contains all oriented neighbors of e which have oriented neighbors outside of $ON(e)$.

<u>Case 1:</u> $Ext(e) = ON(e) - \{e\}$. Set $C = \{e\}$.

<u>Case 2:</u> $Ext(e) \subset ON(e) - \{e\}$. Let $D^0 = ON(e) - Ext(e)$. For $j \geq 0$, while D^j is not a clique let K^j be a maximal clique in D^j and define $D^{j+1} = D^j - K^j$. Let D^k, $k \geq 0$ be the final clique and set $C = D^k$.

It is straightforward to verify that in each of the two cases C is indeed a happy clique.■
□

In the next section we describe an algorithm that will find an oriented edge e such that $r(e)$ is safe given the representation of $OV(\pi)$ described in Section 3.1. The algorithm first finds a happy clique C and then searches for the vertex with maximum unoriented degree in C. According to Theorem 4.1 this vertex defines a safe reversal.

Even though Theorem 4.2 guarantees the existence of a happy clique in the neighborhood of any fixed oriented vertex, our algorithm does not search in one particular such neighborhood. We will prove that the algorithm is guaranteed to find a happy clique assuming that there exists at least one oriented edge. Therefore the algorithm provides an alternative proof to a weaker version of Theorem 4.2 that only claims the existence of a happy clique somewhere in the graph.

4.1. Finding a happy clique. In this section we give an algorithm that locates a happy clique in $OV(\pi)$. Let $e_1, \ldots, e_k$ be the oriented vertices in $OV(\pi)$ in increasing left endpoint order. The algorithm traverses the oriented vertices in $OV(\pi)$ according to this order. Let $L(e)$ and $R(e)$ be the left and right endpoints, respectively, of vertex e in the realization of $OV(\pi)$. After traversing $e_1, \ldots, e_i$, $1 \leq i \leq k$, the algorithm maintains a happy clique C_i in the subgraph of $OV(\pi)$ induced by these vertices. Assume $|C_i| = j$, $j \leq i$ and let $e_{i_1}, \ldots, e_{i_j}$ be the vertices in C_i where $i_1 < i_2 < \ldots < i_j$. The vertices of C_i are maintained in a linked list ordered in increasing left endpoint order. If there exists an interval that contains all the intervals in C_i then the algorithm maintains a minimal such interval t_i. The clique C_i and the vertex t_i (if exists) satisfy the following invariant.

INVARIANT 4.1.
1) Every vertex $e_l \notin C_i$, $l \leq i$, such that $L(e_{i_1}) < L(e_l)$ must be adjacent to t_i, i.e., $R(e_l) > R(t_i)$.
2) Every vertex $e_l \notin C_i$, $L(e_l) < L(e_{i_1})$ that is adjacent to a vertex in C_i is either adjacent to an interval e_p such that $R(e_p) < L(e_{i_1})$ or adjacent to t_i.

The fact that C_i is happy in the subgraph induced by $e_1, \ldots, e_i$ follows from this invariant. We initialize the algorithm by setting $C_1 = \{e_1\}$. Initially, t_1 is not defined. Let the current interval be e_{i+1}. If $R(e_{i_j}) < L(e_{i+1})$ then C_i is guaranteed to be happy in $OV(\pi)$ since all remaining oriented vertices are not adjacent to C_i. Hence the algorithm stops and returns C_i as the answer. See Figure 4.1(a).

We now assume that $L(e_{i+1}) \leq R(e_{i_j})$ and show how to obtain C_{i+1} and t_{i+1}. We have to consider the following cases.
Case 1. The interval t_i is defined and $R(t_i) < R(e_{i+1})$. Continue with $C_{i+1} = C_i$ and $t_{i+1} = t_i$. See Figure 4.1(b).
Case 2. The interval t_i is not defined or $R(e_{i+1}) \leq R(t_i)$.
a) $R(e_{i_j}) < R(e_{i+1})$ and $L(e_{i+1}) \leq R(e_{i_1})$. C_{i+1} is obtained by adding e_{i+1} to C_i and $t_{i+1} = t_i$. See Figure 4.1(c).
b) $R(e_{i_j}) < R(e_{i+1})$ and $L(e_{i+1}) > R(e_{i_1})$. The clique C_{i+1} consists of e_{i+1} alone and $t_{i+1} = t_i$. See Figure 4.1(d).
c) $R(e_{i+1}) < R(e_{i_j})$. As in the previous case $C_{i+1} = \{e_{i+1}\}$. In this case t_{i+1} is set to e_{i_j}, the last interval in C_i. See Figure 4.1(e).

The following theorem proves that the algorithm above produces a happy clique.

THEOREM 4.3. *Let C_l be the current clique when the algorithm stops. Then C_l is a happy clique in $OV(\pi)$.*

Proof. A straightforward induction on the number of oriented vertices traversed by the algorithm proves that C_l and t_l satisfy Invariant 4.1.

The algorithm stops either when $R(e_{i_j}) < L(e_{l+1})$ or when l is equal to the number of oriented vertices. In either case since C_l is happy in the subgraph induced by $e_1, \ldots, e_l$ it must be happy in $OV(\pi)$. □

The running time of the algorithm is proportional to the number of oriented vertices traversed since a constant amount of work is performed for each such vertex.

4.2. Searching the happy clique. After locating a happy clique C in $OV(\pi)$ we need to search it for a vertex with a maximum number of unoriented neighbors. In this section we give an algorithm that performs this task.

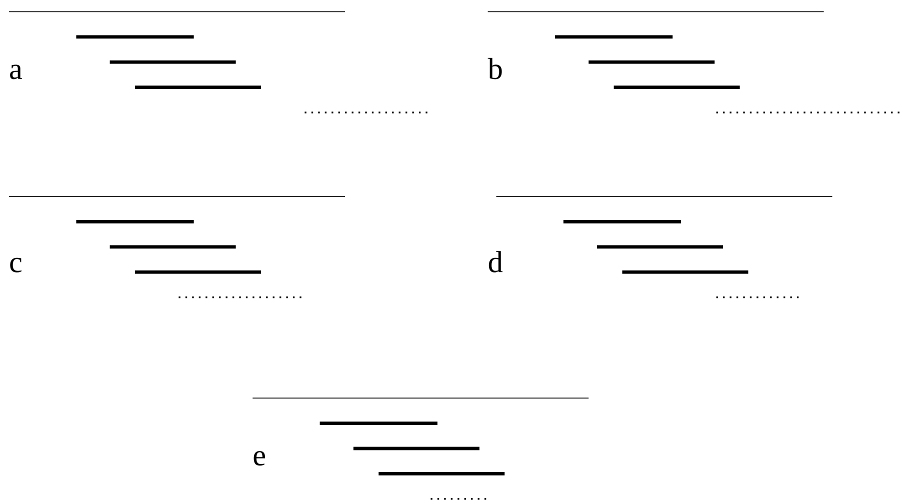

FIG. 4.1. *The various cases of the algorithm for finding a happy clique. The topmost interval is always t_i. The three thick intervals comprise C_i. The dotted interval corresponds to e_{i+1}.*

Let $e_1, \ldots, e_j$ be the intervals in C ordered in increasing left endpoint order. Clearly, $L(1) < L(2) < \ldots < L(j) < R(1) < R(2) < \ldots < R(j)$. Thus the endpoints of the j vertices in C partition the line into $2j+1$ disjoint intervals $I_0, \ldots, I_{2j}$, where $I_0 = (-\infty, L(1)]$, $I_l = (L(l), L(l+1)]$ for $1 \leq l < j$, $I_j = (L(j), R(1)]$, $I_l = (R(l-j), R(l-j+1)]$ for $j < l < 2j$ and $I_{2j} = (R(j), \infty)$. The algorithm consists of the following three stages.

Stage 1: Let e be an unoriented vertex that has a non-empty intersection with the interval $[L(1), R(j)]$. Mark each of e's endpoints with the index of the interval that contains it.

Stage 2: Let o be an array of j counters, each corresponding to a vertex in C. The intention is to assign values to o such that the sum $\sum_{i=1}^{l} o[i]$ is the unoriented degree of the vertex $e_l \in C$. The counters are initialized to zero. For each unoriented vertex e that overlaps with the interval $[L(1), R(j)]$ we change at most four of the counters as follows. Let I_l and I_r be the intervals in which $L(e)$ and $R(e)$ occur, respectively. We may assume $l < r$ as otherwise e is not adjacent to any vertex in C and we can ignore it. We continue according to one of the following cases.

Case 1: $r \leq j$. All the vertices from e_{l+1} to e_r are adjacent to e: we increment $o[l+1]$ and decrement $o[r+1]$ (if $r < j$).

Case 2: $j \leq l$. All the vertices from e_{l-j+1} to e_{r-j} are adjacent to e: we increment $o[l-j+1]$ and decrement $o[r-j+1]$ (if $r < 2j$).

Case 3: $l < j$ and $j < r$. Let $m = \min\{l, r-j\}$. If $m > 0$ then all the vertices from e_1 to e_m are adjacent to e: we increment $o[1]$ and decrement $o[m+1]$. Similarly let $M = \max\{l, r-j\}$. If $M < j$ then the vertices from e_{l+1} to e_j are adjacent to e: we increment the counter $o[l+1]$.

Stage 3: Compute $f = \max_l \{\sum_{i=1}^{l} o[i] | 1 \leq l \leq j\}$. Return e_f.

The following theorem summarizes the result of this section. We omit the proof, which is straightforward.

THEOREM 4.4. *Given a clique C, the vertex $e_f \in C$ computed by the algorithm above has maximum unoriented degree among the vertices in C.*

The complexity of the algorithm is proportional to the size of C plus the number of unoriented vertices in $OV(\pi)$, and hence is $O(n)$.

5. Clearing the hurdles. In case there are unoriented components in $OV(\pi)$, there exists a sequence $r_1, \ldots r_t$ of t reversals that transform π into π' such that $d(\pi') = d(\pi) - t$, where $t = \lceil h(\pi)/2 \rceil$. In this section we summarize the characterization given by Hannenhalli and Pevzner for these t reversals and outline how to find them using our implicit representation of $OV(\pi)$.

We will use the following definitions. A reversal *merges* hurdles H_1 and H_2 if it acts on two breakpoints, one incident with a gray edge in H_1 and the other incident with a gray edge in H_2. Recall the circle CR defined in Section 2, in which the endpoints of the edges in the unoriented components of $OV(\pi)$ are ordered consistently with their order in π. Two hurdles H_1 and H_2 are *consecutive* if their sets of endpoints $E(H_1)$ and $E(H_2)$ occur consecutively on CR, i.e., there is no hurdle H such that $E(H)$ separates $E(H_1)$ and $E(H_2)$ on CR.

The following lemmas were essentially proved by Hannenhalli and Pevzner though stated differently in their paper.

LEMMA 5.1 ([11]). *Let π be a permutation with an even number, say $2k$, of hurdles. Any sequence of $k-1$ reversals each of which merges two non-consecutive hurdles followed by a reversal merging the remaining two hurdles will transform π into π' such that $d(\pi') = d(\pi) - k$ and π' has only oriented components.*

LEMMA 5.2 ([11]). *Let π be a permutation with an odd number, say $2k+1$, of hurdles. If at least one hurdle H is simple then a reversal acting on two breakpoints incident with edges in H transforms π into π' with $2k$ hurdles such that $d(\pi') = d(\pi) - 1$. If π is a fortress then a sequence of $k-1$ reversals merging pairs of non-consecutive hurdles followed by two additional merges of pairs of consecutive hurdles (one merges two original hurdles and the next merges a hurdle created by the first and the last original hurdle) will transform π into π' such that $d(\pi') = d(\pi) - (k+1)$ and π' has only oriented components.*

We now outline how to turn these lemmas into an algorithm that finds a particular sequence of reversals $r_1, \ldots, r_t$ with the properties described above. First $OV(\pi)$ is decomposed into connected components as described in [4]. One then has to identify those unoriented components that are hurdles. This task can be done by traversing the endpoints of the circle CR, counting the number of elements in each run of consecutive endpoints belonging to the same component. If a run contains all endpoints of a particular unoriented component M then M is an hurdle.

In a similar fashion one can check for each hurdle whether it is a simple hurdle or a super hurdle. While traversing the cycle, a list of the hurdles in the order they occur on CR is created. At the next stage this list is used to identify correct hurdles to merge.

We assume that given an endpoint one can locate its connected component in constant time. It is easy to verify that the data can be maintained so that this is possible.

THEOREM 5.3. *Given $OV(\pi)$ decomposed into its connected components, the algorithm outlined above finds t reversals such that when we apply them to π we obtain a π' which is hurdle-free and has $d(\pi') = d(\pi) - t$. The algorithm can be implemented to run in $O(n)$ time.*

Proof. Correctness follows from Lemma 5.1 and 5.2. The time bound is achieved if we always merge hurdles that are separated by a single hurdle. If the ith merge merged hurdles H_1 and H_2 that are separated by H, then H should be merged in the

$i + 1$st merge. Carrying out the merges this way guarantees that the span of each hurdle H overlaps at most two merging reversals, the second of which eliminates H. $\square$

6. Summary. Figure 6.1 gives a schematic description of the algorithm.

algorithm SIGNED REVERSALS(π);
/* π is a signed permutation */
1. Compute the connected components of $OV(\pi)$.
2. Clear the hurdles.
3. while π is not sorted **do** :
/* iteration */
begin
a. find a happy clique C in $OV(\pi)$.
b. find a vertex $e_f \in C$ with maximum unoriented degree, and perform a safe reversal on e_f;
c. update π and the representation of $OV(\pi)$.
end
end 4. output the sequence of reversals.

FIG. 6.1. *An algorithm for sorting signed permutations*

THEOREM 6.1. *Algorithm* SIGNED REVERSALS *finds the reversal distance r in $O(n\alpha(n) + r \times n)$ time, and in particular in $O(n^2)$ time.*

Proof. The correctness of the algorithm follows from Theorem 2.3, Theorem 4.1 and Lemmas 5.1 and 5.2.

Step 1 takes $O(n\alpha(n))$ time by the algorithm of Berman and Hannenhalli [4]. Step 2 takes $O(n)$ time by Theorem 5.3. Step 3 takes $O(n)$ time per reversal, by the discussion in Section 4. $\square$

It is an intriguing open question whether a faster algorithm for sorting signed permutations by reversals exists. It certainly might be the case that one can find an optimal sequence of reversals faster. To date, no nontrivial lower bound is known for this problem.

Acknowledgments. We thank Donald Knuth, Sridhar Hannenhalli, Pavel Pevzner, and Itsik Pe'er for their comments on a preliminary version of this paper.

REFERENCES

[1] W. ACKERMANN, *Zum hilbertshen aufbau der reelen zahlen*, Math. Ann., 99 (1928), pp. 118–133.
[2] V. BAFNA AND P. PEVZNER, *Sorting permutations by transpositions*, in Proceedings of the 6th Annual Symposium on Discrete Algorithms, ACM Press, Jan. 1995, pp. 614–623.
[3] V. BAFNA AND P. A. PEVZNER, *Genome rearragements and sorting by reversals*, SIAM Journal on Computing, 25 (1996), pp. 272–289. A preliminary version appeared in Proc. 34th IEEE Symp. of the Foundations of Computer Science, pages 148–157, 1994.
[4] P. BERMAN AND S. HANNENHALLI, *Fast sorting by reversals*, in Proc. Combinatorial Pattern Matching (CPM), 1996, pp. 168–185. LNCS 1075.
[5] A. CAPRARA, *Sorting by reversals is difficult*, in Proceedings of the First International Conference on Computational Molecular Biology (RECOMB), ACM Press, 1997, pp. 75–83.
[6] D. A. CHRISTIE, *A 3/2-approximation algorithm for sorting by reversals*, in Proc. ninth annual ACM-SIAM Symp. on Discrete Algorithms (SODA 98), ACM Press, 1998, pp. 244–252.
[7] T. DOBZHANSKY AND A. H. STURTEVANT, *Inversions in the chromosomes of drosophila pseudoobscura*, Genetics, 23 (1938), pp. 28–64.

[8] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[9] S. Hannenhalli, *Polynomial algorithm for computing translocation distance between genomes*, Disc. Appl. Math., 71 (1996), pp. 137–151.

[10] S. Hannenhalli and P. Pevzner, *Transforming men into mice (polynomial algorithm for genomic distance problems*, in Proc. IEEE Symp. of the Foundations of Computer Science, 1995, pp. 581–592.

[11] S. Hannenhalli and P. A. Pevzner, *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals)*, in Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, Las Vegas, Nevada, 29 May–1 June 1995, pp. 178–189.

[12] S. B. Hoot and J. D. Palmer, *Structural rearrangements, including parallel inversions, within the chloroplast genome of Anemone and related genera*, J. Molecular Evooution, 38 (1994), pp. 274–281.

[13] H. Kaplan, R. Shamir, and R. E. Tarjan, *Faster and simpler algorithm for sorting signed permutations by reversals*, in Proc. 8th ACM-SIAM Symposium on Discrete Algorithms, ACM-SIAM, 1997, pp. 344–351. Also in *Proc. RECOMB 97*, page 163.

[14] J. Kececioglu and R. Ravi, *Physical mapping of chromosomes using unique probes*, in Proc. sixth annual ACM-SIAM Symp. on Discrete Algorithms (SODA 95), ACM Press, 1995, pp. 604–613.

[15] J. Kececioglu and D. Sankoff, *Efficient bounds for oriented chromosome inversion distance*, in Proc. of 5th Ann. Symp. on Combinatorial Pattern Matching, Springer, 1994, pp. 307–325. LNCS 807.

[16] ———, *Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement*, Algorithmica, 13 (1995), pp. 180–210. A preliminary version appeared in *Proc. CPM93*, Springer, Berlin, 1993, pages 87–105.

[17] J. D. Palmer and L. A. Herbon, *Tricircular mitochondrial genomes of Brassica and Raphanus: reversal of repeat configurations by inversion*, Nucleic Acids Research, 14 (1986), pp. 9755–9764.

[18] ———, *Unicircular structure of the Brassica hirta mitochondrial genome*, Current Genetics, 11 (1987), pp. 565–570.

[19] ———, *Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence*, J. Molecular Evolution, 28 (1988), pp. 87–97.

[20] J. D. Palmer, B. Osorio, and W. Thompson, *Evolutionalry significance of inversions in legume chorloplast DNAs*, Current Genetics, 14 (1988), pp. 65–74.

[21] D. Sankoff, *Edit distance for genome comparison based on non-local operations*, Lecture Notes in Computer Science, 644 (1992), pp. 121–135.

[22] D. Sankoff, R. Cedergren, and Y. Abel, *Genomic divergence through gene rearrangement.*, Methods in Enzymology, 183 (1990), pp. 428–438.

[23] R. E. Tarjan, *Efficiency of a good but not linear set union algorithm*, J. ACM, 22 (1979), pp. 215–225.