World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:6, 2009

On Reversal and Transposition Medians

Martin Bader

Abstract—During the last years, the genomes of more and more species have been sequenced, providing data for phylogenetic reconstruction based on genome rearrangement measures. A main task in all phylogenetic reconstruction algorithms is to solve the median of three problem. Although this problem is NP-hard even for the simplest distance measures, there are exact algorithms for the breakpoint median and the reversal median that are fast enough for practical use. In this paper, this approach is extended to the transposition median as well as to the weighted reversal and transposition median. Although there is no exact polynomial algorithm known even for the pairwise distances, we will show that it is in most cases possible to solve these problems exactly within reasonable time by using a branch and bound algorithm.

Keywords—Comparative genomics, genome rearrangements, median, reversals, transpositions.

I. Introduction

Due to the increasing amount of sequenced genomes, the problem of reconstructing phylogenetic trees based on this data is of great interest in computational biology. In the context of genome rearrangements, a genome is usually represented as a permutation of $(1, \ldots, n)$, where each element represents a gene, i.e. the permutation represents the shuffled ordering of the genes on the genome. Additionally, the strandedness of the genes is taken into account by giving each element an orientation. In the multiple genome rearrangement problem, one searches for a phylogenetic tree describing the most "plausible" rearrangement scenario for multiple genomes. Formally, given k genomes and a distance measure d, find a tree T with the k genomes as leaf nodes and assign ancestral genomes to internal nodes of T such that the tree is optimal w.r.t. d, i.e. the sum of rearrangement distances over all edges of the tree is minimal. If $k = 3$, i.e. one searches for an ancestor such that the sum of the distances from this ancestor to three given genomes is minimized, we speak of the *median problem*. All of the actual state-of-the-art algorithms for solving the multiple genome rearrangement problem rely on algorithms for solving the median problem. Unfortunately, this problem is NP-hard even for the simplest rearrangement measures, namely the *breakpoint distance* [13] and the *reversal distance* [9]. Currently, the most interesting distance measures are:

- The *reversal distance* between two genomes is the minimum number of reversals required to transform one genome into the other. It can be computed in linear time [2]. The reversal median problem has been proven to be NP-hard [9]. The currently best software tools to solve the multiple genome rearrangement problem based on this distance measure are `GRAPPA` [12], `MGR` [8], `amGRP` [7], and `phylo` [3]. While `amGRP` and `phylo` rely on Caprara's median solver [9], `GRAPPA` can alternatively use Siepel's median solver [14]. `MGR` uses its own heuristic median solver.
- The *transposition distance* between two genomes is the minimum number of transpositions required to transform one genome into the other. So far, it is not clear whether it is in P or not, and the currently best approximation algorithm has an approximation ratio of 1.375 [11]. An exact branch and bound algorithm is described in [10]. To the best of our knowledge, the only program that solves the multiple genome rearrangement problem based on this distance measure is `GRAPPA-TP` [18], which uses an extension of Siepel's median solver [14] and solves pairwise distances by a fast heuristic.
- The *weighted reversal and transposition distance* between two genomes is the minimum weight of a sequence consisting of reversals and transpositions that transforms one genome into the other, where reversals and transpositions are weighted differently. Again, it is not clear whether it is in P or not, but there is a 1.5-approximation algorithm that covers each weight ratio from 1:1 to 1:2 (reversals:transpositions) [4]. As far as we know, the only program that solves the multiple genome rearrangement problem based on this distance measure is `phylo` [3], which uses a preliminary version of the median solver presented in this paper.
- The *double cut and join distance* between two genomes is the minimum number of double cut and join (DCJ) operations required to transform one genome into the other. The DCJ operation has been introduced by Yancopoulos et al. [17] and can simulate reversals (with one DCJ operation) and block interchanges (with two DCJ operations), which are a generalization of tranpositions. The DCJ distance can easily be extended to multichromosomal genomes, but it has the drawback of the fixed weight ratio between reversals and block interchanges, and the fact that block interchanges are biologically less motivated than transpositions. While the DCJ distance can be computed in linear time [17], the DCJ median is NP-hard for both the unichromosomal and the multichromosomal case [9], [15]. There are currently several implementations of a DCJ median solver (e.g. [1], [16], [19]) which can be integrated in `MGR` or `GRAPPA`.

In this paper, we will show how one can solve the transposition median as well as the weighted reversal and transposition median by extending Caprara's median solver. This requires the calculation of pairwise distances between two genomes, which either can be done approximately using the algorithm devised in [4], or exactly using a new branch and bound algorithm presented in this paper. Experimental results show that the approximation rate of the first method is very good in

M. Bader is with the Institute of Theoretical Computer Science, Ulm University in Germany, email: martin.bader@uni-ulm.de

practice, and that even the exact algorithm runs in feasible time for practical use. In Section II basic definitions are given. The algorithm to calculate exact pairwise distances is described in Section III, the algorithm to solve the median problem is described in Section IV. The experimental results and a comparison with GRAPPA-TP, which was kindly provided by Jijun Tang, can be found in Section V. Section VI summarizes the method and the results and discusses some future prospects.

II. Preliminaries

A *signed permutation* $\pi = (\pi_1 \dots \pi_n)$ is a permutation of $(1 \dots n)$, where each element π_i has an orientation (indicated by $\overrightarrow{\pi_i}$ or $\overleftarrow{\pi_i}$). In the following, the term "permutation" will be used as short hand for signed permutation. The permutation $id = (\overrightarrow{1} \dots \overrightarrow{n})$ is called the *identity permutation* of size n. A *segment* of a permutation π is a consecutive sequence of elements in π. A *reversal* is an operation that inverts the order of the elements of a segment in a permutation. Additionally, the orientation of every element in the segment is flipped. A *transposition* is an operation that cuts a segment out of a permutation, and reinserts it at another position in the permutation. If additionally a reversal is applied on this segment, we speak of an *inverted transposition*. The *weight* of an operation *op* is denoted by $w(op)$, and the weight of a sequence of operations is the sum of the weights of the operations in the sequence. In the following, reversals have weight w_r, whereas transpositions and inverted transpositions have weight w_t, and it is assumed that $w_r \leq w_t \leq 2w_r$ (otherwise optimal sequences would have an unrealistic strong bias either towards reversals or transpositions). The problem of *sorting by weighted reversals and transpositions* is defined as follows. Given two permutations π^1, π^2, find a sequence of reversals and transpositions of minimum weight that transforms π^1 into π^2. This minimum weight is called the *weighted reversal and transposition distance* (wRTD) $d_w(\pi^1, \pi^2)$. If the set of operations is restricted to transpositions only, the problem is called *sorting by transpositions*, and the corresponding distance is called the *transposition distance* (TD) $d_t(\pi^1, \pi^2)$. Since a transposition can never change the orientation of an element, all the elements in π^1 as well as in π^2 must have positive orientation. Given q permutations $\pi^1, \dots, \pi^q$, the *weighted reversal and transposition median problem* (wRTMP) calls for a permutation ρ such that $\delta(\rho) = \sum_{k=1}^{q} d_w(\rho, \pi^k)$ is minimized. The *transposition median problem* (TMP) is defined analogously. For solving wRTMP and TMP, the *multiple breakpoint graph* is used, which has been introduced by Caprara [9] and is a generalization of the breakpoint graph defined in [5]. For permutations $\pi^1, \dots, \pi^q$, the MB graph $G = (V, E)$ is a multigraph with node set $V = \{-1, +1, -2, +2, \dots, -n, +n\}$ (where n is the size of the permutations). The edge set can be obtained as follows. First, replace in each permutation π^k $(1 \leq k \leq q)$ all elements with positive orientation $\overrightarrow{x}$ by $-x\ +x$ and all elements with negative orientation $\overleftarrow{x}$ by $+x\ -x$. Then, each permutation π^k induces the edge set $M^k = \{(i, j) \mid i \neq -j$ and π^k contains the adjacent values i and $j\ \}$, i.e. the edge set M^k corresponds to the adjacencies in π^k. The edge set E of the MB graph G is the union of these edge sets, i.e. $E = \bigcup_{k=1}^{q} M^k$. As each node is connected to exactly one edge in each edge set M^k, the graphs $G_{i,j} = (V, M^i \cup M^j)$ (with $1 \leq i, j \leq q$) decompose into cycles with alternating edges from the edge sets M^i and M^j. A cycle is called an *odd cycle* if its number of edges divided by 2 is an odd number, otherwise it is called an *even cycle*. Let $c_{odd}(\pi^i, \pi^j)$ denote the number of odd cycles in $G_{i,j}$, and let $c_{even}(\pi^i, \pi^j)$ denote the number of even cycles in $G_{i,j}$. The *score* σ between two permutations π^i and π^j is defined by $\sigma(\pi^i, \pi^j) = c_{odd}(\pi^i, \pi^j) + (2 - \frac{2w_r}{w_t})c_{even}(\pi^i, \pi^j)$. The following theorems show how this score can be used to obtain lower and upper bounds for the wRTD.

Theorem 1: [4], [6] A lower bound $lb_w(\pi^i, \pi^j)$ for the weighted reversal and transposition distance $d_w(\pi^i, \pi^j)$ can be defined as follows.

$$d_w(\pi^i, \pi^j) \geq lb_w(\pi^i, \pi^j),$$
$$\text{where } lb_w(\pi^i, \pi^j) := (n - \sigma(\pi^i, \pi^j))\frac{w_t}{2}$$

A lower bound $lb_t(\pi^i, \pi^j)$ for the transposition distance $d_t(\pi^i, \pi^j)$ can be defined as follows.

$$d_t(\pi^i, \pi^j) \geq lb_t(\pi^i, \pi^j),$$
$$\text{where } lb_t(\pi^i, \pi^j) := (n - c_{odd}(\pi^i, \pi^j)\frac{w_t}{2}$$

Note that if $w_t = 2w_r$, the lower bounds for both distances are equal. This will later simplify the description of the algorithms, as only the lower bound for the wRTD will be used.

Theorem 2: [4], [11] An upper bound $ub_w(\pi^i, \pi^j)$ for the weighted reversal and transposition distance $d_w(\pi^i, \pi^j)$ can be defined as follows.

$$d_w(\pi^i, \pi^j) \leq ub_w(\pi^i, \pi^j),$$
$$\text{where } lb_w(\pi^i, \pi^j) := 1.5lb_w(\pi^i, \pi^j)$$

An upper bound $ub_t(\pi^i, \pi^j)$ for the transposition distance $d_t(\pi^i, \pi^j)$ can be defined as follows.

$$d_t(\pi^i, \pi^j) \leq ub_t(\pi^i, \pi^j),$$
$$\text{where } ub_t(\pi^i, \pi^j) := 1.375lb_t(\pi^i, \pi^j)$$

III. Calculating pairwise distances

As exact polynomial algorithms are neither known for the TD nor for the wRTD, we introduce a branch and bound algorithm for the pairwise distances. The main idea of the algorithm is straightforward. W.l.o.g., the task is to find an optimal sorting sequence between a permutation π and the identity permutation id of the same size. For this, a set S that contains triples $(\tilde{\pi}, d'(\pi, \tilde{\pi}), lb(\tilde{\pi}, id))$ is created, where $\tilde{\pi}$ is a permutation, $d'(\pi, \tilde{\pi})$ is the sum of the weights of all operations that have been performed on the path from π to $\tilde{\pi}$, and $lb(\tilde{\pi}, id)$ is the lower bound for the remaining distance towards id according to Theorem 1. Initially, S is set to $\{(\pi, 0, lb(\pi, id))\}$. In each step, one selects the triple $(\tilde{\pi}, d'(\pi, \tilde{\pi}), lb(\tilde{\pi}, id))$ from S where $d'(\pi, \tilde{\pi}) + lb(\tilde{\pi}, id)$ is minimized, and remove it from S. If $lb(\tilde{\pi}, id) = 0$, then $\tilde{\pi} = id$

and $d'(\pi, \tilde{\pi}) = d(\pi, id)$, i.e. an optimal solution is found and the algorithm aborts. The sequence of operations can be reconstructed by a traceback. Otherwise, for each operation op, the triple $(op\ \tilde{\pi}, d'(\pi, \tilde{\pi}) + w(op), lb(op\ \tilde{\pi}, id)$ is added to S, i.e. all possible predecessors of $\tilde{\pi}$ are added to S. This step is called *expanding* $\tilde{\pi}$. The algorithm continues by again selecting the best triple.

So far, the algorithm is just an ordinary branch and bound algorithm, and does not perform very well in practice. Thus, the algorithm is improved by a duplicate elimination. Because there are usually different optimal sequences to reach an intermediate permutation, this permutation would be stored several times, and in the worst case the number of duplicates of a permutation can be exponential in the distance to the origin permutation. Therefore, it is first checked whether a permutation already has been reached on another sequence before a new triple containing this permutation is created. Searching for a possible duplicate can be done quite efficiently by hashing techniques. The number of elements in S can be further decreased by working on the minimal permutations, which have been defined in [10] as follows. Given a permutation $\tilde{\pi}$, the *minimal permutation* $gl(\tilde{\pi})$ is obtained by 'gluing' all the adjacencies together, i.e. each segment of elements that is identical in $\tilde{\pi}$ and id is replaced by a single element. As an example, the permutations $\tilde{\pi} = (\overrightarrow{1}\ \overrightarrow{2}\ \overrightarrow{4}\ \overrightarrow{3})$ and $\hat{\pi} = (\overrightarrow{1}\ \overrightarrow{3}\ \overrightarrow{4}\ \overrightarrow{2})$ have both the same minimal permutation $(\overrightarrow{1}\ \overrightarrow{3}\ \overrightarrow{2})$. The following lemma ensures that it is sufficient to search for an optimal sorting sequence between $gl(\tilde{\pi})$ and id' to obtain an optimal sorting sequence between $\tilde{\pi}$ and id, where id' is the identity permutation of same size as $gl(\tilde{\pi})$.

Lemma 1: [10] Let π be a permutation and $gl(\pi)$ be its minimal permutation. Let id be the identity permutation of same size as π, and let id' be the identity permutation of same size as $gl(\pi)$. Then, an optimal sorting sequence between $gl(\pi)$ and id' can easily be transformed into an optimal sorting sequence between π and id. Both sorting sequences have the same weight, i.e. $d(\pi, id) = d(\tilde{\pi}, id')$.

Note that the original lemma in [10] only considered the TD. However, the proof for the wRTD works analogously, thus this lemma holds for the TD as well as for the wRTD. While Christie used this proof only to show that one never has to split adjacencies, we will also use it for duplicate elimination. In the example above, $\hat{\pi}$ would be considered to be a duplicate of $\tilde{\pi}$. In fact, instead of storing the original permutations, only the minimal permutations will be stored, resulting in a further space improvement.

IV. The median solver

Our median solver is an extension of Caprara's reversal median solver [9]. While Caprara's algorithm solves instances of the *Cycle Median Problem* (CMP) and reestimates the distances using the reversal distance, we extend the CMP to the *weighted Cycle Median Problem* and reestimate the distances using the TD or the wRTD.

For a given wRTMP instance with permutations $\pi^1, \ldots, \pi^q$, and an arbitrary permutation ρ, define $\gamma(\rho) := \sum_{k=1}^{q} \sigma(\rho, \pi^k)$. The *weighted Cycle Median Problem* (wCMP) is defined as follows. Given a set of q permutations $\pi^1, \ldots, \pi^q$, find a permutation τ such that $qn - \gamma(\tau)$ is minimized. In the following, let ρ^* be the solution of a given wRTMP and let $\delta^* := \delta(\rho^*) = \sum_{i=1}^{q} d_w(\pi^i, \rho^*)$ be its solution value. Let τ^* be the solution of the associated wCPM and let $qn - \gamma^* := qn - \gamma(\tau^*)$ be its solution value. The following lemma shows the relation between a wRTMP instance and the associated wCMP instance.

Lemma 2: Given a wRTMP instance with solution value δ^* and the associated wCMP instance with solution value $qn - \gamma^*$,

$$\frac{w_t}{2}(qn - \gamma^*) \leq \delta^* \leq \frac{3w_t}{4}(qn - \gamma^*)$$

Proof: Using the bounds given in Theorems 1 and 2, we get

$$\begin{aligned}
\frac{w_t}{2}(qn - \gamma^*) &= \frac{w_t}{2}(qn - \gamma(\tau^*)) \\
&\leq \frac{w_t}{2}(qn - \gamma(\rho^*)) \\
&= \sum_{k=1}^{q} lb(\pi^k, \rho^*) \\
&\leq \delta^* \\
&\leq \sum_{k=1}^{q} d(\pi^k, \tau^*) \\
&\leq 1.5 \sum_{k=1}^{q} lb(\pi^k, \tau^*) \\
&= \frac{3w_t}{4}(qn - \gamma^*).
\end{aligned}$$

■

Note that this proof also holds for the TD if $w_r = 1, w_t = 2$, and the search space of the wCMP is restricted to permutations where all elements have positive orientation. In this case, $\gamma(\tau) = \sum_{k=1}^{q} c_{odd}(\pi^k, \tau)$, i.e. an optimal solution of the wCPM maximizes the number of odd cycles. In most cases, δ^* is very close to the lower bound. This motivates the idea to solve a wRTMP instance by solving the associated wCMP instance and then check whether the solution of the wCMP instance is also a solution of the wRTMP instance. We will now address the problem of solving a wCMP instance. As we will use a branch and bound algorithm that successively extends a partial solution until we have a complete solution, the MB graph must be extended such that it can be used to obtain strong lower bounds for partial solutions. A graph (V, E) is *weighted* if each edge $e \in E$ has an integer weight $w(e)$. Given a weighted graph $G = (V, E)$ with node set $V = \{-1, +1, -2, +2, \ldots, -n, +n\}$, a *weighted matching* M is a set of edges in G such that each node in V is incident to at most one edge in M and each edge in M has an odd weight (restricting the weights to be odd will simplify later proofs). A weighted matching M is called *perfect* if each node in V is incident to exactly one edge in M. It is easy to see that the union of two matchings decomposes the graph into *cycles* and *paths* consisting of alternating edges from both matchings. The *length* of a cycle or path is the sum of the weights of its edges. A cycle is called an *odd cycle* if its length divided by 2 is an odd number, otherwise it is called an

even cycle. Note that cycles always consist of an even number of edges, all having an odd weight (recall the definition of weighted matchings), thus the length of a cycle is always divisible by 2. Analogous to the definition given in Section II, $c_{odd}(M^i, M^j)$ is the number of odd cycles in $(V, M^i \cup M^j)$, $c_{even}(M^i, M^j)$ is the number of even cycles in $(V, M^i \cup M^j)$, and $\sigma(M^i, M^j) := c_{odd}(M^i, M^j) + (2 - \frac{2w_r}{w_t})c_{even}(M^i, M^j)$. The *base matching* H is defined by $H := \{(-k, +k) \mid 1 \leq k \leq n\}$ and $\forall e \in H : w(e) = 1$. A weighted matching M is called a *permutation matching* if $H \cup M$ defines a Hamiltonian cycle on G, i.e. a cycle that visits each node in V exactly once.

Lemma 3: [5] There is a one-to-one correspondence between signed permutations and permutation matchings where each edge has weight 1.

In other words, each permutation matching can be transformed into a permutation by ignoring the weights. On the other hand, the search space can be reduced to permutation matchings. Interpreting the MB graph as the special case of a weighted graph (where each weight is set to 1) leads to the following formulation of the wCMP. Given a node set V with $|V| = 2n$ and q permutation matchings $M^1, \ldots, M^q$, find a permutation matching M^τ with edge weights 1 that minimizes $\sum_{k=1}^{q}(n - \sigma(M^\tau, M^k))$. Note that there is no restriction for the weights of the edges of the given permutation matchings. While all edges in the initial problem have weight 1, the branch and bound algorithm will create partial solutions where also other edge weights are possible.

Lemma 4: The *weighted cycle distance* $n - \sigma(S, T)$ on permutation matchings is a metric.

Proof:

1) Positive definiteness: $n - \sigma(S, S) = 0$, because the graph decomposes into n odd cycles. For permutation matchings S, T with $S \neq T$, there must be at least one cycle with at least four edges, thus the overall number of cycles is less than n. As each cycle adds at most 1 to $\sigma(S, T)$, $\sigma(S, T) < n$ and $n - \sigma(S, T) > 0$.
2) Symmetry: This follows directly from the symmetry of $\sigma(S, T)$.
3) Triangle inequation: We show that for permutation matchings S, T, and R, $n - \sigma(S, R) + n - \sigma(R, T) \geq n - \sigma(S, T)$. For this, R is modified successively by the following rules. (a) If $(V, S \cup R)$ contains an even cycle with only two edges, change the weight of the corresponding edge in R such that the cycle becomes odd. This increases $\sigma(S, R)$ by $2\frac{w_r}{w_t} - 1$. In $(V, R \cup T)$, this either changes an even cycle into an odd cycle, or an odd cycle into an even cycle. Thus, $\sigma(S, R) + \sigma(R, T)$ does not decrease. (b) If $(V, S \cup R)$ contains a cycle with at least four edges, remove two of the edges of R and rejoin the endpoints such that the cycle is split into two cycles. Weight the new edges such that both cycles are odd cycles. If the original cycle was even, $\sigma(S, R)$ increases by $\frac{2wr}{wt}$. As the operation can effect at most two cycles in $(V, R \cup T)$, the worst possible effect on $\sigma(R, T)$ is that two odd cycles are merged into an even cycle. Thus, $\sigma(S, R) + \sigma(R, T)$ does not decrease. If the original cycle was odd, $\sigma(S, R)$ increases by 1, and the overall number of odd cycles changes by 1. As the parity of the number of odd cycles is always equal in $(V, S \cup R)$ and $(V, R \cup T)$, the worst possible effect on $\sigma(R, T)$ is that two odd cycles are merged into one odd cycle. Thus, $\sigma(S, R) + \sigma(R, T)$ does not decrease. (c) If none of the two rules above can be applied, S and R contain the same edges, but maybe with different weights. Change the weights of the edges of R such that they have the same weights as the edges in S. Note that this step has no effect on the cycles, as all cycles in $(V, S \cup R)$ are already odd. Thus, $\sigma(S, R) + \sigma(R, T)$ remains unchanged. The whole transformation transformed R into S without decreasing $\sigma(S, R) + \sigma(R, T)$. Therefore, $n - \sigma(S, R) + n - \sigma(R, T) \geq n - \sigma(S, S) + n - \sigma(S, T) = n - \sigma(S, T)$.

■

The following lemma will give us a lower bound for the solution value of a wCMP.

Lemma 5: Given a wCMP instance associated with weighted matchings $M^1, \ldots, M^q$ and solution M^τ, we have

$$\sum_{k=1}^{q}(n - \sigma(M^\tau, M^k)) \geq \frac{qn}{2} - \sum_{k=1}^{q-1}\sum_{l=k+1}^{q}\frac{\sigma(M^k, M^l)}{q-1}$$

Proof: Using the triangle inequality given in Lemma 4, we get

$$\begin{aligned} & \frac{qn}{2} - \sum_{k=1}^{q-1}\sum_{l=k+1}^{q}\frac{\sigma(M^k, M^l)}{q-1} \\ = \; & \frac{1}{q-1}\sum_{k=1}^{q-1}\sum_{l=k+1}^{q}(n - \sigma(M^k, M^l)) \\ \leq \; & \sum_{k=1}^{q}(n - \sigma(M^\tau, M^k)) \end{aligned}$$

■

In order to describe partial solutions, we must introduce the *contraction* of an edge. Given a weighted graph $G = (V, E)$ with $|V| = 2n$ and $E = \bigcup_{k=1}^{q} M^k$, where each M^k is a permutation matching, the *contraction* of an edge $e = (v_i, v_j)$ is an operation that modifies G as follows. The nodes v_i, v_j are removed from V. Each permutation matching M^k is transformed into M^k/e by the following rules. If $e \in M^k$, remove e from M^k, i.e. $M^k/e = M^k \setminus \{e\}$. Otherwise, let (a, v_i) and (b, v_j) be the two edges incident to v_i and v_j in M^k. Remove these edges and add a new edge (a, b), i.e. $M^k/e = M^k \setminus \{(a, v_i), (b, v_j)\} \cup \{(a, b)\}$. The weight of the new edge (a, b) will be set to $w(a, b) := w(a, v_i) + w(b, v_j) + 1$. Note that this is also an odd number, as $w(a, v_i)$ and $w(b, v_j)$ are odd. Analogously, the base matching H will be replaced by H/e.

Lemma 6: [9] Given two perfect matchings M and L of V and an edge $(v_i, v_j) \in M$, $M \cup L$ defines a Hamiltonian cycle of V if and only if $(M/e) \cup (L/e)$ defines a Hamiltonian cycle of $V \setminus \{v_i, v_j\}$.

Lemma 7: Let $M^1, \ldots, M^q$ be a wCMP instance, let M^τ be a permutation matching with edge weights 1, and let $e \in$

M^τ be an edge. Then,

$$\begin{aligned}&\sum_{k=1}^{q}(n-\sigma(M^\tau,M^k))\\ =\;& q-\sum_{k=1}^{q}\sigma(M^k,\{e\})+\sum_{k=1}^{q}(n-1-\sigma(M^\tau/e,M^k/e))\end{aligned}$$

Proof: A cycle in $M^\tau \cup M^k$ is either absorbed by the contraction step, or it corresponds to a cycle in $M^\tau/e \cup M^k/e$ of the same length. In the first case, the absorbed cycle is equivalent to the cycle in $M^k \cup \{e\}$, and the sum of the scores of the absorbed cycles is $\sum_{k=1}^{q}\sigma(M^k,\{e\})$. As there are no new cycles in $M^\tau/e \cup M^k/e$, we get

$$\begin{aligned}&\sum_{i=1}^{q}\sigma(M^\tau,M^k)\\ =\;& -\sum_{k=1}^{q}\sigma(M^k,\{e\})+\sum_{k=1}^{q}(n-\sigma(M^\tau/e,M^k/e))\\ =\;& q-\sum_{k=1}^{q}\sigma(M^k,\{e\})+\sum_{k=1}^{q}(n-1-\sigma(M^\tau/e,M^k/e)).\end{aligned}$$

■

By combining Lemmas 6 and 7, we get the following

Corollary 1: Given a wCMP instance $M^1,\dots M^q$ with solution M^τ and an edge $e \in M^\tau$, then $M^\tau \setminus \{e\}$ is a solution of the wCMP instance $M^1/e,\dots,M^q/e$.

We are now ready to describe our branch and bound algorithm for wCMP. A partial solution consists of a matching that is not necessarily perfect, and the lower bound of a partial solution M can be calculated by contracting all edges in M and calculating the lower bound of the contracted graph, using the formulas described in Lemmas 5 and 7. In each step, the partial solution M with the currently least lower bound is selected and expanded as follows. Let V' be the nodes of V such that forall $v_i \in V', v_j \in V : (v_i,v_j) \notin M$, and let v_a be a fixed node in V'. Then, a new partial solutions M' is created by setting $M' = M \cup (v_a,v_b)$ for all $v_b \in V', v_b \neq v_a$. Partial solutions M' that cannot be expanded to a permutation matching (i.e. $M' \cup H$ contains a cycle that is not a Hamiltonian cycle) can be directly discarded, for all other partial solutions the lower bounds are calculated. The algorithm has found an optimal solution for the wCMP when the partial solution with the least lower bound is a perfect matching.

The algorithm can easily be extended such that it can solve the wRTMP or the TMP by adding the following step. Whenever the best partial solution M^τ is a perfect matching, create the corresponding permutation π^τ and test if $\sum_{k=1}^{q} d_w(\pi^k,\pi^\tau)$ is equal to the lower bound (of course one has to take d_t instead of d_w if one wants to solve the TMP). In this case, an optimal solution is found. Otherwise, the lower bound for M^τ is increased, and M^τ is reinserted into the set of partial solutions. A further speed-up of the pairwise distance algorithm can be obtained by providing an upper bound (remember that we only want to test if the sum of the pairwise distances is equal to the lower bound, thus the pairwise distance algorithms can be aborted if the currently best results are above this bound). If one wants to solve the TMP, partial solutions are further restricted to matchings where all edges are of the form $(+i,-j)$, because other edges correspond to a change in orientation in the permutation and therefore these permutations cannot be sorted by transpositions only.

V. Experimental results

The algorithm has been tested on artificial datasets with 37 and 100 markers, reflecting the size of mitochondrian and chloroplast genomes. The datasets were created by, starting from the identity permutation, creating three different sequences of operations to get the input genomes. The weight of the edges is uniformly distributed in $[0.5r, 1.5r]$, where r is the expected weight of an edge, varying from 2 to 15. For the data sets to test the transposition median solver, w_t was set to 1. For the data sets to test the weighted reversal and transposition median solver, w_r was set to 1, and different datasets were created for $w_t = 1$, $w_t = 1.5$, and $w_t = 2$. When creating the data sets, the probability of performing a transposition reflects the weight of w_t, i.e. the expected ratio of reversals to transpositions is $w_t : w_r$. For each combination of these parameters, we created 10 data sets. All tests were performed on a standard 3.16 GHz PC, the running time for each test case was limited to one hour, and RAM was limited to 4GB.

The experiments showed that the size of the datasets has only little influence on the results. Up to an expected edge length of $r = 8$, all test cases could be solved exactly, most of them even in less than one second. For higher distances, the running times increased. However, we could still solve 9 instances of the TMP with $r = 15$ and $n = 37$ with an average running time of 6:21 min, and 6 instances of the TMP with $r = 15$ and $n = 100$ with an average running time of 16:27 min. For the instances of the wRTMP, the running times depend on the used weight ratio. While setting $w_r : w_t$ to $1 : 2$ allowed us to solve all test cases within a few seconds, the running times increased for the other weight ratio. Moreover, the heap had to be pruned due to the memory limit in some cases, which means that there is a slight chance that wethe optimal solution has been missed. Nevertheless, we were still able to solve all test cases, except for a few test cases with $n = 100$, $w_r : w_t = 1 : 1$, and $r > 13$. Using the approximation algorithm instead of the exact algorithm for the pairwise distances resulted in better running times at almost the same accuracy. In most cases, a solution of same weight as with the exact algorithm was found, and the gap between the weights of the solutions never exceeded 1.

A comparison with `GRAPPA-TP` on the instances of the TMP shows that `GRAPPA-TP` is slightly less accurate than our median solver, but its main drawback is the speed. For $r = 7$, its average running time was 3:36 min ($n = 37$) respectively 6:41 min ($n = 100$), while our algorithm solved these test cases within less than one second. Increasing the edge lengths further decreased the number of solved test cases. For $n = 37$, none of the test cases with $r \geq 14$ could be solved within one hour. For $n = 100$, none of the test cases with $r \geq 11$ could be solved within one hour.

A more detailed view of the test results can be found in

Appendix A. In the tables, the number of solved test cases and the average running time of the approximation algorithm and the exact algorithm is shown for each combination of parameters, as well as the average gap and the maximum gap between the solution of the approximation algorithm and the exact algorithm. Of course, the gaps can only be computed for test cases which have been solved by both algorithms, and the average running times only consider test cases that could be solved within the time limit. The column "proven exact" indicates the number of test cases where it can be assured that the exact algorithm did not miss an optimal solution due to heap pruning. Note that the other solutions might still be exact. In fact, as only the currently worst solutions are pruned, the probability of missing an optimal solution is rather low.

VI. Conclusion and Future Work

We presented an extension of Caprara's median solver that can solve instances of the TMP and the wRTMP. The method has been tested on artificial datasets, showing that is possible to solve the wRTMP and the TMP exactly in many cases. A comparison with `GRAPPA-TP` on TMPs shows that our algorithm brings a speed improvement of several orders of magnitude. However, there is still room for improvements, like e.g. a graph decomposition approach similar to the one in [16]. As this approach gave an immense speedup for the DCJ median problem, we expect similar results for the wRTMP. The program is free software and can be obtained from the authors.

References

[1] Z. Adam and D. Sankoff. The ABCs of MGR with DCJ. *Evolutionary Bioinformatics*, 4:69–74, 2008.

[2] D. Bader, B. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8:483–491, 2001.

[3] M. Bader, M. Abouelhoda, and E. Ohlebusch. A fast algorithm for the multiple genome rearrangement problem with weighted reversals and transpositions. *BMC Bioinformatics*, 9:516, 2008.

[4] M. Bader and E. Ohlebusch. Sorting by weighted reversals, transpositions, and inverted transpositions. *Journal of Computational Biology*, 14(5):615–636, 2007.

[5] V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.

[6] V. Bafna and P. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.

[7] M. Bernt, D. Merkle, and M. Middendorf. Using median sets for inferring phylogenetic trees. *Bioinformatics*, 23:e129–e135, 2007.

[8] B. Bourque and P. Pevzner. Genome-scale evolution: Reconstructing gene orders in the ancestral species. *Genome Research*, 12(1):26–36, 2002.

[9] A. Caprara. The reversal median problem. *INFORMS Journal on Computing*, 15(1):93–113, 2003.

[10] D. Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, 1998.

[11] I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379, 2006.

[12] B. Moret, S. Wyman, D. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *Proc. 6th Pacific Symposium on Biocomputing*, pages 583–594, 2001.

[13] I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. *Electronic Colloquium on Computational Complexity*, 5(71), 1998.

[14] A. Siepel and B. Moret. Finding an optimal inversion median: Experimental results. In *Proc. 1st Workshop on Algorithms*, volume 2149 of *Lecture Notes in Computer Science*, pages 189–203. Springer-Verlag, 2001.

[15] E. Tannier, C. Zheng, and D. Sankoff. Multichromosomal genome median and halving problems. In *Proc. 8th Workshop on Algorithms in Bioinformatics*, volume 5251 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2008.

[16] A. Xu. A fast and exact algorithm for the median of three problem - a graph decomposition approach. In *Proc. 6th Annual RECOMB Satellite Workshop on Comparative Genomics*, volume 5267 of *Lecture Notes in Bioinformatics*, pages 184–197. Springer-Verlag, 2008.

[17] S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

[18] F. Yue, M. Zhang, and J. Tang. Phylogenetic reconstruction from transpositions. *BMC Genomics*, 9(Suppl 2):S15, 2008.

[19] M. Zhang, W. Arndt, and J. Tang. An exact solver for the DCJ median problem. In *Proc. 14th Pacific Symposium on Biocomputing*, pages 138–149. World Scientific, 2009.

APPENDIX A
DETAILED EXPERIMENTAL RESULTS

r	solved app	avg gap	max gap	avg time	solved exact	proven exact	avg time
2	10	0	0	0:00	10	10	0:00
3	10	0	0	0:00	10	10	0:00
4	10	0	0	0:00	10	10	0:00
5	10	0	0	0:00	10	10	0:00
6	10	0	0	0:00	10	10	0:00
7	10	0	0	0:06	10	9	1:16
8	10	0	0	0:02	10	10	0:04
9	10	0	0	0:15	10	9	2:22
10	10	0.22	1	1:17	9	5	6:00
11	10	0	0	2:12	8	3	1:54
12	10	0.11	1	2:18	9	2	7:22
13	10	0.11	1	1:59	8	3	4:17
14	10	0.13	1	4:37	8	0	15:32
15	10	0.11	1	2:57	9	3	6:21

TABLE I
$n = 37$, TRANSPOSITION DISTANCE

r	solved app	avg gap	max gap	avg time	solved exact	proven exact	avg time
2	10	0	0	0:00	10	10	0:00
3	10	0	0	0:00	10	10	0:00
4	10	0	0	0:00	10	10	0:00
5	10	0	0	0:00	10	10	0:00
6	10	0	0	0:00	10	10	0:00
7	10	0	0	0:00	10	10	0:00
8	10	0	0	0:00	10	10	0:00
9	10	0	0	0:00	10	10	0:00
10	10	0	0	0:00	9	9	0:00
11	10	0	0	0:04	10	10	0:04
12	9	0	0	0:25	9	8	0:30
13	9	0.17	1	7:50	6	4	2:35
14	9	0.14	1	2:54	7	5	0:19
15	7	0.17	1	7:24	6	3	16:27

TABLE II
$n = 100$, TRANSPOSITION DISTANCE

r	solved app	avg gap	max gap	avg time	solved exact	proven exact	avg time
2	10	0	0	0:00	10	10	0:00
3	10	0	0	0:00	10	10	0:00
4	10	0	0	0:00	10	10	0:00
5	10	0	0	0:00	10	10	0:00
6	10	0.1	1	0:00	10	10	0:00
7	10	0	0	0:01	10	10	0:01
8	10	0	0	1:20	10	8	1:24
9	10	0.1	1	0:06	10	10	0:03
10	10	0.2	1	2:48	10	6	3:01
11	10	0	0	4:19	10	2	6:17
12	10	0.2	1	5:23	10	2	6:04
13	10	0.3	1	9:38	10	1	11:05
14	10	0.2	1	9:28	10	0	13:18
15	10	0.3	1	10:49	10	1	16:21

TABLE III
$n = 37$, $w_r = 1$, $w_t = 1$

r	solved app	avg gap	max gap	avg time	solved exact	proven exact	avg time
2	10	0	0	0:00	10	10	0:00
3	10	0	0	0:00	10	10	0:00
4	10	0	0	0:00	10	10	0:00
5	10	0	0	0:00	10	10	0:00
6	10	0	0	0:00	10	10	0:00
7	10	0	0	0:00	10	10	0:00
8	10	0	0	0:05	10	10	0:05
9	9	0	0	1:01	10	7	1:22
10	10	0	0	0:38	10	8	0:52
11	10	0	0	4:21	10	7	4:49
12	10	0	0	0:53	10	7	0:58
13	8	0	0	2:09	8	6	2:39
14	7	0	0	16:46	7	4	19:33
15	4	0	0	13:40	3	2	1:30

TABLE IV
$n = 100$, $w_r = 1$, $w_t = 1$

r	solved app	avg gap	max gap	avg time	solved exact	proven exact	avg time
2	10	0	0	0:00	10	10	0:00
3	10	0	0	0:00	10	10	0:00
4	10	0	0	0:00	10	10	0:00
5	10	0	0	0:00	10	10	0:00
6	10	0.05	0.5	0:00	10	10	0:00
7	10	0.05	0.5	0:00	10	10	0:00
8	10	0.05	0.5	0:00	10	10	0:00
9	10	0.05	0.5	0:00	10	10	0:00
10	10	0.15	0.5	0:06	10	9	0:01
11	10	0	0	0:10	10	9	0:11
12	10	0.1	0.5	0:05	10	8	0:05
13	10	0	0	0:56	10	8	1:16
14	10	0.1	0.5	1:09	10	7	1:33
15	10	0.05	0.5	0:09	10	8	0:09

TABLE V
$n = 37$, $w_r = 1$, $w_t = 1.5$

r	solved app	avg gap	max gap	avg time	solved exact	proven exact	avg time
2	10	0	0	0:00	10	10	0:00
3	10	0	0	0:00	10	10	0:00
4	10	0	0	0:00	10	10	0:00
5	10	0	0	0:00	10	10	0:00
6	10	0.05	0.5	0:00	10	10	0:00
7	10	0.05	0.5	0:00	10	10	0:00
8	10	0	0	0:00	10	10	0:00
9	10	0.05	0.5	0:02	10	10	0:00
10	10	0	0	0:00	10	10	0:00
11	10	0.1	0.5	0:00	10	10	0:00
12	10	0	0	0:00	10	10	0:00
13	10	0.2	1	0:01	10	10	0:01
14	10	0.05	0.5	0:07	10	10	0:03
15	10	0	0	0:20	10	8	0:19

TABLE VI
$n = 100$, $w_r = 1$, $w_t = 1.5$

r	solved app	avg gap	max gap	avg time	solved exact	proven exact	avg time
2	10	0	0	0:00	10	10	0:00
3	10	0	0	0:00	10	10	0:00
4	10	0	0	0:00	10	10	0:00
5	10	0	0	0:00	10	10	0:00
6	10	0.1	1	0:00	10	10	0:00
7	10	0	0	0:00	10	10	0:00
8	10	0	0	0:00	10	10	0:00
9	10	0	0	0:00	10	10	0:00
10	10	0	0	0:00	10	10	0:00
11	10	0	0	0:00	10	10	0:00
12	10	0	0	0:00	10	10	0:00
13	10	0	0	0:00	10	10	0:00
14	10	0	0	0:02	10	10	0:03
15	10	0	0	0:01	10	10	0:01

TABLE VII
$n = 37$, $w_r = 1$, $w_t = 2$

r	solved app	avg gap	max gap	avg time	solved exact	proven exact	avg time
2	10	0	0	0:00	10	10	0:00
3	10	0	0	0:00	10	10	0:00
4	10	0	0	0:00	10	10	0:00
5	10	0	0	0:00	10	10	0:00
6	10	0.1	1	0:00	10	10	0:00
7	10	0	0	0:00	10	10	0:00
8	10	0	0	0:00	10	10	0:00
9	10	0.1	1	0:00	10	10	0:00
10	10	0	0	0:00	10	10	0:00
11	10	0	0	0:00	10	10	0:00
12	10	0	0	0:00	10	10	0:00
13	10	0	0	0:00	10	10	0:00
14	10	0	0	0:00	10	10	0:00
15	10	0	0	0:00	10	10	0:00

TABLE VIII
$n = 100$, $w_r = 1$, $w_t = 2$

r	solved	avg gap	max gap	avg time
2	10	0	0	0:00
3	10	0.2	2	0:04
4	10	0.3	2	0:16
5	9	0.22	1	0:22
6	8	0.13	1	2:57
7	8	0.5	2	3:36
8	1	0	0	13:25
9	3	0.33	1	4:23
10	3	0.2	3	0:21
11	0			
12	0			
13	0			
14	0			
15	0			

r	solved	avg gap	max gap	avg time
2	10	0	0	0:00
3	10	0	0	0:05
4	10	0.1	1	0:29
5	10	0	0	0:34
6	10	0.4	2	2:06
7	10	0.2	1	6:41
8	10	0	0	12:35
9	7	0.43	2	19:08
10	1	1	1	16:16
11	3	0	0	7:54
12	2	0	0	21:34
13	1	1	1	39:55
14	0			
15	0			

TABLE IX
GRAPPA-TP, $n = 37$ (LEFT) AND $n = 100$ (RIGHT).