A Unified Integer Programming Model for Genome Rearrangement Problems

Giuseppe Lancia, Franca Rinaldi, and Paolo Serafini

Dipartimento di Matematica e Informatica.
University of Udine,
Via delle Scienze 206, (33100) Udine, Italy.

Abstract. We describe an integer programming (IP) model that can be applied to the solution of all genome-rearrangement problems in the literature. No direct IP model for such problems had ever been proposed prior to this work. Our model employs an exponential number of variables, but it can be solved by column generation techniques. I.e., we start with a small number of variables and we show how the correct missing variables can be added to the model in polynomial time.

Keywords: Genome rearrangements; Evolutionary distance; Sorting by reversals; Sorting by transpositions; Pancake flipping problem

1 Introduction

With the large amount of genomic data available today it is now possible to try and compare the genomes of different species, in order to find their differences and similarities. The model of sequence alignment is inappropriate for genome comparisons, where differences should be measured not in terms of insertions/deletions or mutations of single nucleotides, but rather of macroscopic events, affecting long genomic regions at once, that have happened in the course of evolution. These events occur mainly in the production of sperm and egg cells (but also for environmental reasons), and have the effect of rearranging the genetic material of parents in their offspring. When such mutations are not lethal, after a few generations they can become stable in a population and give rise to the birth of new species.

Among the main evolutionary events known, some affect a single chromosome (e.g., *inversions*, and *transpositions*), while others exchange genomic regions from different chromosomes (e.g., *translocations*). In this paper we will focus on the former type of evolutionary events. When an inversion or a transposition occurs, the fragment is detached from its original position and then it is reinserted, on the same chromosome. In an inversion, it is reinserted at the same place, but with opposite orientation than it originally had. In a transposition, similarly to a cut-and-paste operation in text editing, the fragment is pasted into a new position. Moreover, the fragment can preserve its original orientation, or it can be reversed (in which case we talk of an *inverted transposition*).

Since evolutionary events affect long DNA regions (several thousand bases), the basic unit for comparison is not the nucleotide, but rather the gene. In fact, the computational study of rearrangement problems started after it was observed that many species share the same genes (i.e., the genes have identical, or nearly identical, DNA sequences), however differently arranged. For example, most genes of the mitochondrial genome of *Brassica oleracea* (cabbage) are identical in *Brassica campestris* (turnip), but appear in a completely different order. Much of the pioneering work in genome rearrangement problems is due to Sankoff and his colleagues [22].

Under the assumption that for any two species s and s' there is a closest common ancestror c in the "tree of life", there exists a set of evolutionary events that, if applied to s, can turn s back into c and then into s'. That is, there exists a path of evolutionary events turning s into s'. The length of this path is correlated to the so-called *evolutionary distance* between s and s'. The general genome comparison problem can then be stated as follows:

> *Given two genomes (i.e., two sets of common genes differently ordered), find which sequence of evolutionary events where applied by Nature to the first genome to turn it into the second.*

A general and widely accepted parsimony principle states that the solution sought is the one requiring the minimum possible number of events (a weighted model, based on the probability of each event, would be more appropriate, but these probabilities are very hard to determine). In the past years people have concentrated on evolution by means of some specific type of event alone, and have shown that these special cases can already be very hard to solve [2, 3, 9, 11, 18]. Only in a very few cases models with more than one event have been considered [1, 20].

The ILP approach. When faced with difficult (i.e., NP-hard) problems, the most successful approach in combinatorial optimization is perhaps the use of Integer Linear Programming (ILP) [21]. In the ILP approach, a problem is modeled by defining a set of integer variables and a set of linear constraints that these variables must satisfy in order to represent a feasible solution. The optimal solution is found by minimizing a linear function over all feasible solutions. The solution of an ILP model can be carried on by resorting to a standard package, such as the state-of-the-art program `CPLEX`. The package solves the ILP via a branch-and-bound procedure employing mathematical programming ideas.

Since the solution algorithm is already taken care of, the main difficulty in the ILP approach is the design of the model, i.e., the definition of proper variables, constraints and objective function. Genome rearrangement problems have thus far defied the attempts of using ILP models for their solutions, since no direct ILP models seemed possible. In this paper we describe a very general new ILP model which can be used for all genome rearrangement problems in the literature.

2 Sorting Permutations

Two genomes are compared by looking at their common genes. After numbering each of n common genes with a unique label in $[n] := \{1, \ldots, n\}$ each genome is a permutation of the elements of $[n]$. Let $\pi = (\pi_1, \ldots, \pi_n)$ and $\tau = (\tau_1, \ldots, \tau_n)$ be two genomes. The generic genome rearrangement problem requires to find a shortest sequence of operators that applied to the starting permutation π yields τ as the final permutation. By possibly relabeling the genes, we can always assume that $\tau = \iota = (1, 2, \ldots, n)$, the identity permutation. Hence, the problem becomes *sorting* π by means of the allowed operators.

Each operator represents an evolutionary event that might happen to π so as to change the order of its genes. The most prominent such events are

1. **Reversal**. A fragment of the permutation is flipped over so that its content appears reversed. For instance, a reversal between positions 3 and 6 applied to
$$(2, 7, \boxed{4,\ 1,\ 5,\ 6}, 3)$$
yields
$$(2, 7, 6, 5, 1, 4, 3)$$
An interesting special case are *prefix reversals*, in which one of the pivoting points is always position 1, and therefore a prefix of the permutation is flipped over. For instance, a prefix reversal until position 4 applied to
$$(\boxed{2,\ 7,\ 4,\ 1}, 5, 6, 3)$$
yields
$$(1, 4, 7, 2, 5, 6, 3)$$
2. **Transposition**. A fragment of the permutation is cut from its original position and pasted into a new position. The operator is specified by 3 indexes i, j, k where i and j specify the starting and ending position of the fragment, and k specifies the position where the fragment will be put (i.e. the fragment will end at position k after its placement). For instance a transposition with $i = 2$, $j = 3$ and $k = 6$ applied to
$$(2, \boxed{7,\ 4}, 1, 6, 5, {}_{\uparrow}\, 3)$$
yields
$$(2, 1, 6, 5, 7, 4, 3)$$
3. **Inverted transposition**. A fragment of the permutation is cut from its original position and pasted into a new position, but with the order of its elements flipped over. Of course this operator can be mimicked by a reversal followed by a transposition (or by a transposition followed by a reversal), but here we consider this as a single move and not two moves. Again, the operator is specified by 3 indexes i, j, k where i and j specify the stating and ending position of the fragment, and k specifies the position where the fragment

will be put (i.e. the fragment will end at position k after its placement). For instance an inverted transposition with $i = 2$, $j = 3$ and $k = 6$ applied to

$$(2, \boxed{7,\ 4}, 1, 6, 5, _{\uparrow}\, 3)$$

yields

$$(2, 1, 6, 5, 4, 7, 3)$$

To each of these operators there corresponds a particular sorting problem. Therefore we have (i) Sorting by Reversals (SBR); (ii) Sorting by Prefix Reversals (also knonw as the *Pancake Flipping Problem*, SBPR); (iii) Sorting by Transpositions (SBT); (iv) Sorting by Inverted Transpositions (SBIT).

Permutation sorting problems are all very challenging and their complexity has for a long time remained open. As for the solving algorithms, there are not may excact approaches for these problems in the literature. In particular, there are not many ILP approaches, since the problems did not appear to have simple, "natural"' ILP formulations.

Historically, the study of permutation sorting problems started with the pancake flipping problem (SBRP), a few years before the age of bioinformatics. This problem has later gained quite some popularity due to the fact that Bill Gates, the founder of `Microsoft`, co-authored the first paper on its study [15]. The complexity of SBPR has remained open for over 30 years and recently the problem was shown to be NP-hard [6]. Papadimitriou and Gates have proved that the diameter of the pancake flipping graph (i.e., the maximum prefix-reversal distance between two permutations) is $\leq \frac{5}{3}n$. This bound has remain unbeated until recently, and we now know that it is $\leq \frac{18}{11}n$ [12]. There is no published algorithm for exact solution of SBPR, an no ILP formulation of the problem. There is an approximation algorithm achieving an approximation factor of 2 [14].

The other permutations sorting problems have started being investigated in the early nineties for their potential use in computing evolutionary distances. A particular interest was put in sorting by reversals. The problem of SBR was shown to be NP-hard by Caprara [8]. The first approximation algorithm for SBR achieves a factor $\frac{3}{2}$ [13], later improved to 1.375 [5]. Integer Linear Programming has been successfully applied to SBR [10, 11], but in an "indirect" way. Namely, the approach exploits an auxiliary maximum cycle-decomposition problem on a bicolored special graph (the *breaklpoint graph* [2]). In this modeling, neither the permutation nor the sorting reversals of the solution are directly seen as variables and/or costraints of the model. Simply, ILP is used for the breakpoint graph problem which yields a tight bound to SBR. The latter is not modeld as an ILP, but solved in a branch-and-bound fashion, while employing the cycle-decomposition bound.

For SBPR and SBT (let alont SBIT) there are not even exact algorithms in the literature. There are approximation algorithms for SBT[3] and SBIT[17, 16]. Only recently SBT has been proven NP-hard [7], closing a longstanding open question.

Results. In this paper we describe an ILP model for genome rearrangement problems. Our ILP is very general and possesses some nice features, such as (i) it yields (for the first time) a model for rearranging under two or more operators (such as prefix reversals *and* transpositions, or reversals *and* transpositions, etc.; (ii) it can be used to model each type of rearrangement separately; (iii) we can easily incorporate limitations on some operators (e.g., we may allow only reversals of regions of some bounded length, or transpositions which cannot move a fragment too far away from its original position); (iv) we can easily incorporate different costs for the various operators.

3 The Basic Model

Assume we have fixed the type of operator (or operators) of the particular sorting problem. Let L be an upper bound to the number of operators to sort π (obtained, e.g., by running a heuristic). We will considered a layered graph G with L layers. In particular, there will by L layers of edges (numbered $1, \ldots, L$), and $L+1$ levels of nodes (numbered $1, \ldots, L+1$). Each level of nodes consists of n nodes, which represent the current permutation. The nodes of level k are meant to represent the permutation after $k-1$ solution steps. Between each pair of consecutive levels there are n^2 arcs (i.e., the two levels form a complete bipartite graph). These arcs are meant to show "where each element goes" (i.e., an arc from node i of level k to node j of level $k+1$ means that the k-th operator has moved an element from i-th position to j-th position).

The nodes of the first level are labeled by the starting permutation π, while the nodes of level $L+1$ are labeled by the sorted permutation. Given a solution, if we apply it to π and follow how elements are moved around by the sequence of operators, we will see that, for each $i = 1, \ldots, n$, the element in position i in π must eventually end in position π_i in the sorted permutation. Hence, the solution individues n node-disjoint directed paths in the layered graph, one for each node i of level 1, ending in nodes π_i on level $L+1$. In Figure 1 we show an example of SBPR, with $L = 5$, and a solution corresponding to the prefix reversals $(< 2 >, < 5 >, < 3 >, < 4 >)$, where by $< i >$ we mean "reverse the first i elements". In bold we can see the path followed by the element labeled 3 in its movements toward its final position.

The model we propose has variables associated to paths in G (it is therefore an exponential model, but we will show how column-generation can be carried out in polynomial time).

Let $\mathcal{P}(i)$ be the set of paths in the layered graph which start at node i of level 1, and end at node π_i of level $L+1$. Note that there are potentially n^{L-1} such paths. From each node, exactly one path must leave, which can be enforced by constraints

$$\sum_{P \in \mathcal{P}(i)} x_P = 1 \qquad \forall i = 1, \ldots, n \tag{1}$$

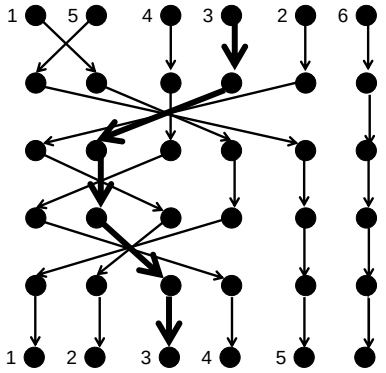

Fig. 1. Solving an instance of SBPR

Furthermore, in going from one level k to the next, the paths should only be allowed along a particular subset of arcs, i.e., the arcs corresponding to the k-th operator used.

Let us consider the complete bipartite graph $[n] \times [n]$, representing all the arcs from a level to the next. Each operator can be seen as a subset of these arcs, in particular, as a special perfect matching. To maintain the generality of the approach, we will just say that the sorting problem is defined by a set of perfect matchings $\mathcal{O}$ in $[n] \times [n]$. Each matching represents a legal operation that can be applied to the current permutation. We will assume that the identity matching $\mu_0 = \{(1,1),(2,2),\ldots,(n,n)\}$ always belongs to $\mathcal{O}$, so that $\mathcal{O} = \{\mu_0, \mu_1, \ldots, \mu_N\}$. The identity permutation specifies a special operation applied to π, i.e., the *no-operation*. It represents a null-event that should be used to fill the "extra" moves that the layered graph can fit with respect to the optimal solution (e.g., if $L = 10$ but the optimal solution uses only 7 moves, then there will be 3 no-operations in the solution using 10 moves).

We now introduce binary variables z_μ^k, for $k = 1, \ldots, L$ and $\mu \in \mathcal{O}$, with the meaning: "in going from level k to level $k+1$ of G, the subset of arcs that can be used by the paths is μ". Otherwise stated, "the k-th move of the solution is the operator (corresponding to) μ".

Since at each level we must use an operator (possibly, the null-operator), we have constraints

$$\sum_{\mu \in \mathcal{O}} z_\mu^k = 1 \qquad \forall k = 1, \ldots, L \tag{2}$$

We can see the setting of z_μ^k to 1 as the *activation* of a particular set of arcs at level k. Then, we have constraints stating that the paths can only use activated arcs, i.e.,

$$\sum_{i=1}^{n} \sum_{P \in \mathcal{P}(i) \,|\, e \in P} x_P \leq \sum_{\mu \in \mathcal{O} \,|\, (u,v) \in \mu} z_\mu^k \qquad \forall e = (u^k, v^{k+1}) \in E \tag{3}$$

where E is the edge set of all arcs in the layered graph, and the generic arc $e \in E$ connects node u^k (with $u \in [n]$) of level k to node v^{k+1} (with $v \in [n]$) of level $k+1$.

Notice that the r.h.s. involves a number $O(|\mathcal{O}|)$ of variables. As we will soon see, $|\mathcal{O}|$ is polynomial for the specific sorting problems of interest in genomics, namely, $|\mathcal{O}| = O(n)$ for SBPR, $|\mathcal{O}| = O(n^2)$ for SBR, $|\mathcal{O}| = O(n^3)$ for SBT and SBIT.

As for the objective function, the parsimony model calls for minimizing the number of non-null operators used, i.e.

$$\min \sum_{k=1}^{L} \sum_{\mu \in \mathcal{O} - \mu_0} z_\mu^k \tag{4}$$

Furthermore, we can add constraints in order to avoid different but equivalent solutions, that place the null operators at various levels of the layered graphs. In particular, we can enforce a "canonical" form of the solution in which all the true operators are at the beginning and then followed by the null operators. We obtain this via the constraint

$$z_{\mu_0}^k \leq z_{\mu_0}^{k+1} \qquad \forall k = 1, \ldots, L-1 \tag{5}$$

The general IP model. Putting all together, we have the following ILP model for the "Sorting by X" problem, where X is a particular type of operator/s. Let $\mathcal{O}$ be the set of all legal moves under the operator/s X:

$$\min \sum_{k=1}^{L} \sum_{\mu \in \mathcal{O} - \mu_0} z_\mu^k \tag{6}$$

subject to

$$\sum_{\mu \in \mathcal{O}} z_\mu^k = 1 \qquad \forall k = 1, \ldots, L \tag{7}$$

$$\sum_{P \in \mathcal{P}(i)} x_P = 1 \qquad \forall i = 1, \ldots, n \tag{8}$$

$$\sum_{i=1}^{n} \sum_{P \in \mathcal{P}(i) \,|\, e \in P} x_P \leq \sum_{\mu \in \mathcal{O} \,|\, (u,v) \in \mu} z_\mu^k \qquad \forall e = (u^k, v^{k+1}) \in E \tag{9}$$

$$z_{\mu_0}^k \leq z_{\mu_0}^{k+1} \qquad \forall k = 1, \ldots, L-1 \tag{10}$$

$$z_\mu^k \in \{0,1\}, \quad x^P \geq 0 \qquad \forall k = 1, \ldots, L, \, \forall \mu \in \mathcal{O}, \, \forall P \in \cup_i \mathcal{P}(i) \tag{11}$$

Notice that only z variables need to be integer, since it can be shown that when the z are integer the x will be as well.

3.1 Solving the Pricing Problem

The above model has an exponential number of path variables. However, its Linear Programming (LP) relaxation can still be solved in polynomial time provided we can show how to solve the *pricing* problem for the path variables. The resulting approach is called *column generation* [4]. The idea is to start with only a subset S of the x variables. Then, given an optimal solution to the current LP(S), we see if there is any missing x variable that could be profitably added to S (or, as usually said in the O.R. community, that could be *priced-in*).

Let $\gamma_1, \ldots, \gamma_n$ be the dual variables associated to constraints (8) and let λ_e, for $e \in E$, be the dual variables associated to constraints (9). To each primal variable y_P corresponds an inequality in the dual LP. The variable should be priced-in if and only if the corresponding dual constraints is violated by the current optimal dual solution. Assume P is a path in $\mathcal{P}(i)$. Then, the corresponding dual inequality for P is

$$\gamma_i - \sum_{e \in P} \lambda_e \leq 0 \tag{12}$$

If we consider λ_e as edge lengths, and define $\lambda(P) := \sum_{e \in P} \lambda_e$, we have that the dual inequalities, for all $P \in \mathcal{P}(i)$, are of type

$$\lambda(P) \geq \gamma_i \tag{13}$$

A path violates the dual inequality if $\lambda(P) < \gamma_i$. Notice that if the shortest path in $\mathcal{P}(i)$ has length $\geq \gamma_i$, then no path in $\mathcal{P}(i)$ can violate inequality (13). Hence, to find a path which violates (13), it is enough to solve n shortest-path problems, one for each $i \in [n]$. Being the graph layered, each shortest path can be found in time $O(m)$, where m is the actual number of arcs in the graph.

By the above discussion, we have in fact proved the following theorem:

Theorem 1. *The LP relaxation of (6)-(11) can be solved in polynomial time.*

4 A Compact Model

Alternatively to the exponential-size model with path variables, we describe a compact reformulation (see [19] for a review about compact optimization) based on a multi-commodity flow (MCF) model. The two models are equivalent, i.e., from a feasible solution of either one of the two we can obtain a feasible solution to the other which has the same objective-function value.

The compact model employs variables x_{ab}^{ki} for each level $k = 1, \ldots, L$, each "source" $i \in [n]$ (representing a node of level 1), and each pair $a, b \in [n]$ representing an arc of layer k (i.e., a is a level-k node and b is a level-$(k+1)$ node). The variable represent the amount of flow started from node i, traveling on the arc (a, b) and going to node π_i on level $L + 1$.

We have flow outgoing constraints:

$$\sum_{j=1}^{n} x_{ij}^{1i} = 1 \qquad \forall i = 1, \ldots, n \tag{14}$$

flow ingoing constraints

$$\sum_{j=1}^{n} x_{j,\pi_i}^{Li} = 1 \qquad \forall i = 1, \ldots, n \tag{15}$$

flow-conservation constraints:

$$\sum_{j=1}^{n} x_{aj}^{ki} - \sum_{j=1}^{n} x_{ja}^{k-1,i} = 0 \qquad \forall a = 1, \ldots, n, \quad \forall k = 2, \ldots, L \tag{16}$$

capacity constraints (i.e., arc activation):

$$\sum_{i=1}^{n} x_{ab}^{ki} \leq \sum_{\mu \in \mathcal{O} \mid (a,b) \in \mu} z_{\mu}^{k} \qquad \forall a, b \in [n], \quad k = 1, \ldots, L \tag{17}$$

The final MCF model is obtained from the path model by replacing constraints (8) and (9) with (14), (15), (16) and (17).

5 The Operators

Let us now describe the sets of perfect matchings corresponding to operators of interst for genomic rearrangement problems. We also add the prefix reversal operator, which, although not intersting in genomics, has received a lot of attention in the literature as the famous Pancake Flipping problem.

Reversals and Prefix Reversals. There are $\binom{n}{2}$ reversals, one for each choice of indexes $1 \leq i < j \leq n$. The generic reversal between positions i and j is identified (see Fig. 2(a)) by the perfect matching

$$\mu(i,j) := \{(1,1), \ldots, (i-1, i-1), (i,j), (i+1, j-1), \ldots, (j,i), (j+1, j+1), \ldots, (n,n)\}$$

Let us call $\mathcal{R}$ the set of all reversals.

When in a reversal $\mu(i,j)$ it is $i = 1$, we talk of *prefix reversal*. The generic prefix reversal up to position j is identified (see Fig. 2(b)) by the perfect matching

$$\mu(j) := \{(1,j), (2, j-1), \ldots, (j,1), (j+1, j+1), \ldots, (n,n)\}$$

Let us call $\tilde{\mathcal{R}}$ the set of all prefix reversals.

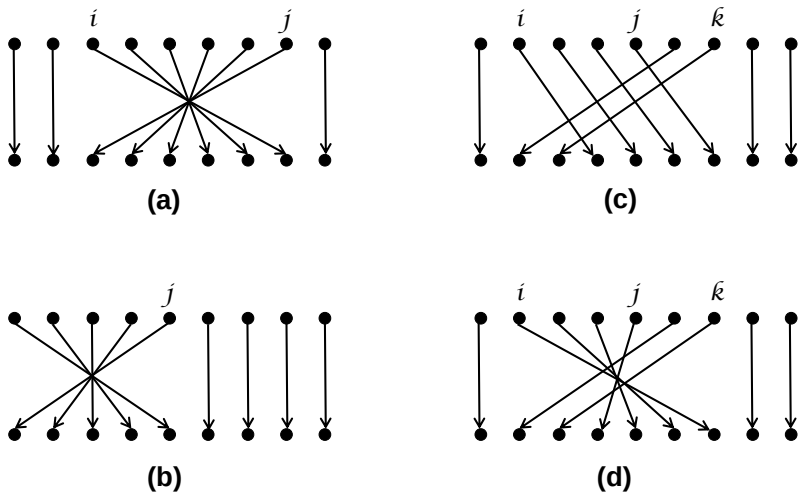

Fig. 2. (a) A reversal; (b) A prefix reversal; (c) A transposition; (d) An inverted transposition.

Transpositions and Inverted Transpositions. Each transposition can be seen as the exchange of two consecutive blocks, where one of the two (but not both) can also have length 1. Let i be the position of the starting of the first block and k the position where the second block ends. Let j denote the position where the first block ends (inclusive). The transposition is identified by the triple

$$1 \leq i \leq j < k \leq n$$

Therefore, there are $\binom{n}{3} + \binom{n}{2}$ transpositions. The generic transposition is identified (see Fig. 2(c)) by the perfect matching

$$\begin{aligned}\mu(i,j,k) := &\{(1,1),\ldots,(i-1,i-1)\} \cup \\ &\{(i,i+k-j),(i+1,i+k-j+1),\ldots,(j,k)\} \cup \\ &\{(j+1,i),(j+2,i+1),\ldots,(k,i+k-j-1)\} \cup \\ &\{(k+1,k+1),\ldots,(n,n)\}\end{aligned}$$

Let us call $\mathcal{T}$ the set of all transpositions. Similarly, an inverted transposition in which the block from i to j is reversed and exchanged with the block from $j+1$ to k corresponds (see Fig.2(d)) to the perfect matching

$$\begin{aligned}\mu'(i,j,k) := &\{(1,1),\ldots,(i-1,i-1)\} \cup \\ &\{(i,k),(i+1,k-1),\ldots,(j,i+k-j)\} \cup \\ &\{(j+1,i),(j+2,i+1),\ldots,(k,i+k-j-1)\} \cup \\ &\{(k+1,k+1),\ldots,(n,n)\}\end{aligned}$$

Let us call $\mathcal{I}$ the set of all inverted transpositions.

The problems that we can model. Given the above definitions, there are many "sorting by X" problems that can be solved by our generic ILP model. The optimization of some of these problems has already been studied in the literature, while for some others (i.e., those considering more than one type of operator) optimization approaches have never been designed due to the problems hardness. In order to model a particular problem, all we need to do is specify which is the allowed set $\mathcal{O}$ of moves from one level to the next. Therefore, here are the problems that we can solve:

- **Sorting by Prefix Reversals (a.k.a. Pancake Flipping):** $\mathcal{O} := \{\mu_0\} \cup \tilde{\mathcal{R}}$.
- **Sorting by Reversals:** $\mathcal{O} := \{\mu_0\} \cup \mathcal{R}$.
- **Sorting by Transpositions:** $\mathcal{O} := \{\mu_0\} \cup \mathcal{T}$.
- **Sorting by Inverted Transpositions:** $\mathcal{O} := \{\mu_0\} \cup \mathcal{I}$.
- **Sorting by Reversals and Transpositions:** $\mathcal{O} := \{\mu_0\} \cup \mathcal{R} \cup \mathcal{T}$.
- **Sorting by Reversals and General Transpositions:**. $\mathcal{O} := \{\mu_0\} \cup \mathcal{R} \cup \mathcal{T} \cup \mathcal{I}$.

Furthermore, we can easily model situations in which some moves are forbidden, e.g., depending on the length of the fragment involved. For instance, if only fragments up to a maximum length of l can be reversed/transposed, (which is reasonable and has been studied in some computational biology papers) this can be easily modeled by redefing the above sets so as to contain only the allowed matchings.

References

1. Bader, M., Ohlebusch, E.: Sorting by weighted reversals, transpositions, and inverted transpositions, J. Comput. Biol. 14, 615-636 (2007)
2. Bafna, V., Pevzner, P.: Genome rearrangements and sorting by reversals, SIAM J. Comp. 25, 272-289 (1996)
3. Bafna, V., Pevzner, P.: Sorting by transpositions. SIAM J. Discr. Math. 11, 224–240 (1998)
4. Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., Vance, P. H.: Branch-and-Price: Column Generation for Solving Huge Integer Programs. Op. Res. 46, 316–329 (1998)
5. Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-Approximation algorithm for sorting by reversals. European Symposium on Algorithms 2002. LNCS, vol. 2461, pp. 200–210, Springer, Heidelberg (2002)
6. Bulteau, L., Fertin, G., Rusu, I.: Pancake Flipping is Hard. In: Rovan, B., Sassone, V., Widmayer, P. (Eds.), International Symposium on Mathematical Foundations of Computer Science 2012. LNCS, vol. 7464, pp. 247–258. Springer, Heidelberg (2012)
7. Bulteau, L., Fertin, G., Rusu, I.: Sorting by Transpositions Is Difficult. SIAM J. Discr. Math. 26, 1148–
8. Caprara, A.: Sorting by reversals is difficult. In: 1st ACM/IEEE International Conference on Computational Molecular Biology, pp. 75–83, ACM Press (1997)
9. Caprara, A.: Sorting Permutations by Reversals and Eulerian Cycle Decompositions. SIAM J. on Disc. Math. 12, 91–110 (1999)

10. Caprara, A., Lancia, G., Ng, S-K.: A Column-Generation Based Branch-and-Bound Algorithm for Sorting By Reversals. In: Mathematical Support For Molecular Biology, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 47, 213-226 (1999)
11. Caprara, Lancia, G., Ng, S.K.: Sorting Permutations by Reversals through Branch and Price, INFORMS J. on Comp. 13, 224-244 (2001)
12. Chitturi, B., Fahle, W., Meng, Z., Morales, L., Shields, C.O., Sudborough, I. H., Voit, W.: An $18/11n$ upper bound for sorting by prefix reversals. Theor. Comp. Sc. 410, 3372–3390 (2009)
13. Christie, A.: A 3/2-approximation algorithm for sorting by reversals. In: 9th ACM-SIAM Symposium on Discrete Algorithms, pp. 244–252, ACM press (1998)
14. Fischer, J., Ginzinger, S.: A 2-Approximation Algorithm for Sorting by Prefix Reversals, European Symposium on Algorithms 2005. LNCS, vol. 3669, pp. 415–425, Springer, Heidelberg (2005)
15. Gates, W., Papadimitriou, C.: Bounds for sorting by prefix reversal. Discr. Math. 27, 47–57 (1979)
16. Gu, Q.P., Peng, S., Sudborough, H.: A 2-approximation algorithm for genome rearrangements by reversals and transpositions. Theoret. Comput. Sci. 210, 327–339 (1999)
17. Hartman, T., Sharan, R.: A 1.5-approximation algorithm for sorting by transpositions and transreversals. J. Comput. Syst. Sci. 70, 300-320 (2005)
18. Kececioglu, J., Sankoff, D.: Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. Algorithmica 13, 180–210 (1995)
19. Lancia, G., Serafini, P.: Deriving compact extended formulations via LP-based separation techniques. 4OR 12, 201–234 (2014)
20. Meidanis, J., Maria M. T. Walter, and Zanoni Dias, A Lower Bound on the Reversal and Transposition Diameter, J. Comput. Biol. 9, 743-745 (2002)
21. Nemhauser, G. L., Wolsey, L. A.: Integer and Combinatorial Optimization. Wiley, 784 pages (1999)
22. Sankoff, D., Cedergren, R., Abel, Y.: Genomic divergence through gene rearrangement. In: Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences, pp. 428–438, Academic Press, New York (1990)