Algorithms for a Restricted Genome Median Problem

Helmuth O. M. Silva[1], Diego P. Rubert[1], Eloi Araujo[1], Fábio V. Martinez[1]

[1]Faculdade de Computação, Universidade Federal de Mato Grosso do Sul
Campo Grande – MS – Brazil

helmuth.silva@ufms.br, {diego,feloi,fhvm}@facom.ufms.br

Abstract. *In the* median problem *we are given a set of three or more genomes and want to find a new genome minimizing the sum of pairwise distances between it and the given genomes. For almost all rearrangement operations the median problem is NP-hard. We study the median problem under a restricted rearrangement measure called* c_4-distance*, which is closely related to breakpoint and DCJ distances. We propose two algorithms for its construction, one exact ILP-based and a combinatorial heuristic, and perform experiments on simulated data.*

1. Background

Almost all known rearrangement distances can be computed efficiently under the assumption that genomic markers appear unique in each genome [Fertin et al. 2009, Yancopoulos et al. 2005, Bergeron et al. 2006]. However, constructing a median of three genomes is NP-hard under almost all rearrangement distances, including *double-cut-and-join* (DCJ) [Tannier et al. 2009], with two notable exceptions: the breakpoint distance and the closely related *single-cut-or-join* (SCJ) are tractable for multichromosomal circular and mixed genomes [Tannier et al. 2009, Feijão and Meidanis 2011].

The *breakpoint graph* supports the computation of rearrangement distances and in doing so, the contained number of cycles plays an essential role. To compute the breakpoint distance, only cycles of length 2 are counted and to compute the DCJ distance cycles of any length are counted. Here we study the c_4-*distance* between two genomes, which is based on the number of cycles of length at most 4. Thus, it is required to count those cycles with at most one DCJ operation to be transformed into adjacencies.

A *marker* is an oriented DNA fragment. We denote a marker either by m or by $\overline{m}$, depending on its orientation. A *chromosome* is a sequence of markers. We consider only circular genomes and use a string of markers to represent a chromosome, adding parentheses at the extremities. A *genome* is a collection of chromosomes. A marker m has two *extremities* represented by m^t (*tail*) and m^h (*head*). We denote an *adjacency* in a chromosome by a pair of consecutive marker extremities. As an example, $(\overline{5}\ 2\ 4\ 3\ 6\ \overline{1})$ is a circular chromosome and $\{(3), (\overline{5}\ 2\ 6), (4\ \overline{1})\}$ is a multichromosomal genome.

One can compute a rearrangement distance between two given genomes with the support of a breakpoint graph [Bafna and Pevzner 1996]. Let $\mathcal{M}$ be a set of n markers and define $\mathcal{M}_x$ the set of extremities of all markers in $\mathcal{M}$, with $|\mathcal{M}_x| = 2n$. For two genomes A and B, each one with n markers from $\mathcal{M}$, the *breakpoint graph* $BG(A, B)$ is the multigraph whose vertex set is $\mathcal{M}_x$ and the edges are of two types: A-edges and B-edges, corresponding to adjacencies in genomes A and B, respectively. This graph has vertices with degree zero, one or two, and thus it is a collection of paths and cycles.

Let $\mathbf{\Pi}$ be a set with $p \geq 3$ genomes, each one with n markers from $\mathcal{M}$. The *median problem* on $\mathbf{\Pi}$ asks for finding a genome Γ with n markers from $\mathcal{M}$ minimizing the pairwise distances between Γ and each genome in $\mathbf{\Pi}$, under a rearrangement operation. If the operation is the SCJ, then the median can be computed in polynomial time [Feijão and Meidanis 2011]. For the DCJ operation, the median problem is NP-hard, even for $p = 3$ [Tannier et al. 2009, Caprara 2003]. Motivated by the searching for where is the boundary of efficiency-hardness of the median problem, we define the *c_4-distance* between Π_i and Π_j, denoted by d_4, given by $\mathrm{d}_4(\Pi_i, \Pi_j) = n - c_2 - c_4$, where n is the number of markers in $\mathcal{M}$, and c_ℓ is the number of ℓ-cycles in $BG(\Pi_i, \Pi_j)$, $\ell \in \{2, 4\}$.

Let $\mathbf{\Pi} = \{\Pi_1, \ldots, \Pi_p\}$ be a set of 3 genomes and Γ be a genome. The *c_4-cost* $K(\mathbf{\Pi}, \Gamma)$ of Γ given $\mathbf{\Pi}$ is $K(\mathbf{\Pi}, \Gamma) = \sum_{\Pi_i \in \mathbf{\Pi}} \mathrm{d}_4(\Pi_i, \Gamma)$. We say that a genome Γ is a *c_4-median* of a set of 3 genomes $\mathbf{\Pi}$ if Γ minimizes the c_4-cost $K(\mathbf{\Pi}, \Gamma)$. For a given set of genomes $\mathbf{\Pi}$, we denote by $K^\star(\mathbf{\Pi})$ the value of its c_4-median: $K^\star(\mathbf{\Pi}) = \min\{K(\mathbf{\Pi}, \Gamma) :$ genomes in $\mathbf{\Pi}$ and genome $\Gamma\}$. Thus, we can state the following:

Problem c_4-MEDIAN($\mathbf{\Pi}$): Given 3 genomes in $\mathbf{\Pi}$, each with n markers from $\mathcal{M}$, find a genome Γ with n markers from $\mathcal{M}$ such that $K^\star(\mathbf{\Pi}) = K(\mathbf{\Pi}, \Gamma)$.

2. ILP

Our exact integer linear program (ILP) algorithm translates the minimization formula for the c_4-median problem in a straightforward way. Suppose we are given a set of three genomes $\mathbf{\Pi} = \{\Pi_1, \Pi_2, \Pi_3\}$ on $\mathcal{M}$, where each genome Π_i is represented by n adjacencies of its markers. Then, we check whether an arbitrary adjacency is an i-colored 2-cycle or whether two arbitrary adjacencies are an i-colored 4-cycle in the "extended" breakpoint graph $BG_x(\mathbf{\Pi})$—a breakpoint graph for more than two genomes—, for each possible adjacency of a pair in $\mathcal{M}_x$. We then maximize these quantities to obtain the solution.

Algorithm 1 ILP for computing the c_4-median

$$
\begin{array}{llcll r}
\text{min} & & & 3n - \sum\limits_{\substack{\pi \in \Pi_i \\ \Pi_i \in \mathbf{\Pi}}} c_{2,i}^{\pi} - \sum\limits_{\substack{\pi, \sigma \in \Pi_i \\ \Pi_i \in \mathbf{\Pi}}} c_{4,i}^{\pi,\sigma} & & \\
\text{s. t.} & \sum\limits_{\substack{v \in \mathcal{M}_x \\ v \neq u}} uv & = & 1 & \forall\, u \in \mathcal{M}_x & \text{(C.01)} \\
& c_{2,i}^{\pi} & = & \pi & \forall\, \pi = uv \in \Pi_i, \forall\, \Pi_i \in \mathbf{\Pi} & \text{(C.02)} \\
& 2c_{4,i}^{\pi,\sigma} & \leq & \begin{array}{l} ux + uy + \\ vx + vy \end{array} & \left.\begin{array}{l} \forall\, \pi = uv \\ \forall\, \sigma = xy \end{array}\right\} \in \Pi_i, \forall\, \Pi_i \in \mathbf{\Pi} & \text{(C.03)} \\
\text{and} & uv & \in & \{0,1\} & \forall\, u, v \in \mathcal{M}_x, u \neq v & \text{(D.01)} \\
& c_{2,i}^{\pi} & \in & \{0,1\} & \forall\, \pi = uv \in \Pi_i, \forall\, \Pi_i \in \mathbf{\Pi} & \text{(D.02)} \\
& c_{4,i}^{\pi,\sigma} & \in & \{0,1\} & \left.\begin{array}{l} \forall\, \pi = uv \\ \forall\, \sigma = xy \end{array}\right\} \in \Pi_i, \forall\, \Pi_i \in \mathbf{\Pi} & \text{(D.03)}
\end{array}
$$

We impose that one and only one egde in the solution is chosen for each possible extremity u in $\mathcal{M}_x$ of a marker in $\mathcal{M}$ (constraint (C.01) and binary variable (D.01)). If an adjacency $\pi = uv$ of a genome Π_i is chosen for extremities $u, v \in \mathcal{M}_x$, then the binary variable $c_{2,i}^{\pi}$ is set to 1. Otherwise, $c_{2,i}^{\pi}$ is set to 0 ((C.02) and (D.02)). Besides that, if we have two adjacencies $\pi = uv$ and $\sigma = xy$ in $\mathcal{M}_x$ and a pair of distinct adjacencies is chosen for extremities u, v, x, y in $\mathcal{M}_x$, then the binary variable $c_{4,i}^{\pi,\sigma}$ is set to 1. Otherwise, $c_{4,i}^{\pi,\sigma}$ is set to 0 ((C.03) and (D.03)). Finally, for genomes Π_i, $1 \leq i \leq 3$, we maximize $c_{2,i}^{\pi}$ and $c_{4,i}^{\pi,\sigma}$ for all possible adjacencies π and σ of extremities of $\mathcal{M}_x$, which corresponds to the objective function. Observe that we have $O(n^2)$ constraints and variables in Algorithm 1.

3. Edge scores heuristic

Let $G = BG_x(\mathbf{\Pi})$ be the extended breakpoint graph of a set $\mathbf{\Pi}$ of given genomes, each one with n markers from the set of markers $\mathcal{M}$. Let $\mathcal{R}$ be the set of *reliable edges*, and an edge e in G belongs to $\mathcal{R}$ if it has multiplicity at least 2. Let Γ be a matching in G. The *score* s of an edge uv in Γ is $s(uv) = t + \frac{1}{2}f$, where t is the number of i-colored 2-cycles and f is the number of i-colored 4-cycles such that uv belongs to, with $i \in \{1, 2, 3\}$. Notice that $0 \leq t + f \leq 3$. Let $s(\Gamma) := \sum_{uv \in \Gamma} s(uv)$. The two edges in Γ of an i-colored 4-cycle in G^Γ are *sibling edges*. Then, for each edge uv in Γ, the *cycle potential* λ in G^Γ is $\lambda(uv) = \frac{1}{2}(\mu(uv)+3) - s(uv)$. The cycle potential $\lambda(uv)$ of an edge uv in Γ represents the possibility of involvement of uv in other 2- and 4-cycles in G^Γ.

Algorithm 2 starts choosing reliable edges and arbitrary remaining edges to be part of an initial solution, obtaining a 1-regular graph Γ with vertex set $V(G)$. Then, it computes score and cycle potential for each edge in Γ. The next step is trying to increase the score of an edge, and to decrease the cycle potential of a small subset of edges, through local changes. It repeats this process while the sum of the scores of all edges increases from one step to the next and there is an edge with positive cycle potential. Observe that the search in line 4, for an edge in G, takes $O(n)$ time and each line from 5 to 11 can be performed in constant time. Moreover, $s(\Gamma)$ can be increased $O(n)$ times, which means that the running time of Algorithm 2 is $O(n^2)$.

Algorithm 2 Edge scores

Input: Genomes in $\mathbf{\Pi}$ and extended breakpoint graph $G = BG_x(\mathbf{\Pi})$
Output: 1-regular graph Γ

```
 1: let $\mathcal{R}$ be the set of reliable edges of $G$
 2: let $\Gamma$ be a 1-regular graph comprised of $\mathcal{R}$ and arbitrary remaining edges
 3: compute $s(uv), \lambda(uv)$ for each edge $uv$ in $\Gamma$
 4: if there exists an edge $uv$ in $\Gamma$ such that $\lambda(uv) > 0$ then
 5:     let $uu_1, vv_1$ be $i$-colored edges, $u_1v_1 \notin \Gamma, i \in \{1, 2, 3\}$
 6:     let $u_1u_2, v_1v_2$ be edges in $\Gamma$
 7:     if $u_1u_2, v_1v_2 \notin \mathcal{R}$ then
 8:         $\Gamma = \Gamma + \{u_1v_1, u_2v_2\} - \{u_1u_2, v_1v_2\}$
 9:         update $s(uv), \lambda(uv)$
10:         update $s$,$\lambda$ for sibling edges of $u_1u_2$,$v_1v_2$,$u_1v_1$,$u_2v_2$
11:         compute $s(u_1v_1), \lambda(u_1v_1)$ and $s(u_2v_2), \lambda(u_2v_2)$
12: repeat lines 4–11 while $s(\Gamma)$ can be increased without removing reliable edges in $\mathcal{R}$
13: return $\Gamma$
```

4. Experiments

We simulated multiple genomes in order to (i) sketch the boundaries where our ILP (Algorithm 1) can perform in reasonable time whilst providing an acceptable accuracy, and (ii) evaluate the quality and running times of our heuristic (Algorithm 2). Experiments were run using 3.60GHz CPUs. We implemented the heuristic in Python 3 and used Gurobi 9.0.2 as ILP solver with 8 cores and time limit of 1 hour.

Given a root genome with n markers, a *descendant trio* is a set of three genomes, each one generated by simulating independently $\frac{n}{100} \cdot k$ random DCJs in the root genome. We generated firstly root genomes with $\{50, 60, \ldots, 400\}$ markers distributed in a couple circular chromosomes and then, for each root genome, five descendant trios varying k in $\{10, 15, 20, 25, 30\}$ (Fig. 1). For this dataset, reaching the time limit seems to depend more on the absolute number of DCJs used to generate the trios than on the genome sizes, as the solver always spent 1 hour after around 30 DCJs (magenta line in Fig. 1(a)). Conversely, the results suggest that the gap between the solution returned by the ILP solver and the best lower bound known grows slower than the running time (Fig. 1(b)). Gaps did not grow over 8% when $k \leq 25$ for most genomes in this dataset (red regions in Fig. 1(b)). The running times of Algorithm 2 were negligible, and the solutions differed 2.8% on average, respectively, from the (not necessarily optimal) solutions given by the solver. Second, in order to stress the heuristic and evaluate their performance, we simulated datasets with large genomes, from 1,000 to 10,000 markers and $k = 30$.

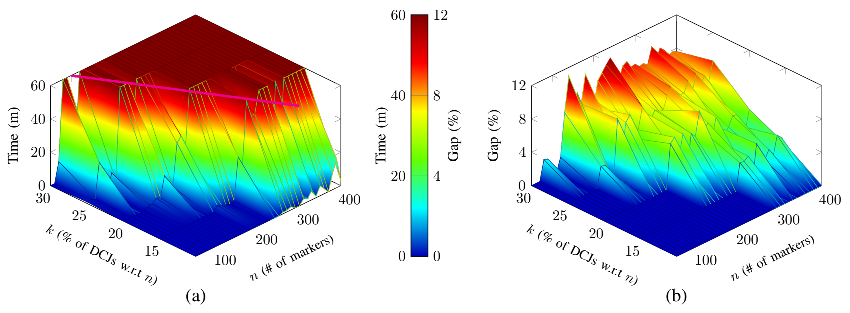

Figure 1. For multiple genome sizes and k values, (a) running times of the ILP solver in minutes and (b) gap between the solution returned by the ILP solver and the best lower bound known.

5. Conclusion

In this work we study the median problem under the c_4-*distance*. We develop algorithms, one exact ILP-based and one combinatorial heuristic, which allow us to perform experiments on simulated data and to provide many insights about the problem. Moreover, this work offers many perspectives for future research. From the theoretical perspective, the computational complexity of the problem is still open. Algorithms proposed give a practical perspective to the problem and allow to compare their results to those for breakpoint and DCJ median. Particularly, we want to design a strategy to speed up Algorithm 1, such as a branch and bound algorithm. On the other hand, we also want to establish approximations factors to the heuristic (Algorithm 2). Besides that, a real data analysis, of single-celled organisms or mitochondrial DNA of more complex organisms, should give us more information about the behaviour of this measure in practice.

References

Bafna, V. and Pevzner, P. A. (1996). Genome rearrangements and sorting by reversals. *SIAM J Comput*, 25(2):272–289.

Bergeron, A., Mixtacki, J., and Stoye, J. (2006). A unifying view of genome rearrangements. In *Proc. of WABI*, volume 4175 of *LNBI*, pages 163–173.

Caprara, A. (2003). The reversal median problem. *INFORMS J Comput*, 15:93—113.

Feijão, P. and Meidanis, J. (2011). SCJ: A breakpoint-like distance that simplifies several rearrangement problems. *IEEE/ACM Trans Comput Biol Bioinf*, 8(5):1318–1329.

Fertin, G., Labarre, A., Rusu, I., Tannier, E., and Vialette, S. (2009). *Combinatorics of Genomes Rearrangements*. The MIT Press.

Tannier, E., Zheng, C., and Sankoff, D. (2009). Multi-chromosomal median and halving problems under different genomic distances. *BMC Bioinformatics*, 10(120).

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