Reactive Stochastic Local Search Algorithms for the Genomic Median Problem

Renaud Lenne[1,2], Christine Solnon[1], Thomas Stützle[2], Eric Tannier[3], and Mauro Birattari[2]

[1] LIRIS, UMR CNRS 5205, Université de Lyon 1, Lyon, France
{renaud.lenne,christine.solnon}@liris.cnrs.fr
[2] IRIDIA-CoDE, Université Libre de Bruxelles, Brussels, Belgium
{stuetzle,mbiro}@ulb.ac.be
[3] INRIA Rhône-Alpes, LBBE, UMR CNRS 5558, Université de Lyon 1, France
eric.tannier@inria.fr

Abstract. The genomic median problem is an optimization problem inspired by a biological issue: it aims at finding the chromosome organization of the common ancestor to multiple living species. It is formulated as the search for a genome that minimizes a rearrangement distance measure among given genomes. Several attempts have been reported for solving this problem. These range from simple heuristic methods to a stochastic local search algorithm inspired by `WalkSAT`, a well-known local search algorithm for the satisfiability problem in propositional logic. The main objective of our research is to develop improved algorithmic techniques for tackling the genomic median problem. In particular, we have developed an algorithm that is based on tabu search and iterated local search. To alleviate the dependence of the algorithm performance on a single fixed parameter setting, we have included a reactive scheme that automatically adapts the tabu list length of the tabu search part and the perturbation strength of the iterated local search part. In fact, computational results show that our final algorithm reaches very high performance for the genomic median problem and we have found a new best solution for a real-world case.

1 Introduction

Genome rearrangements are large-scale evolutionary events that modify the organization of genomes. Chromosomes may be fissioned, fusioned, large segments can be translocated or inverted. Given the genome of living species, the reconstruction of rearrangement scenarios has been the subject of a large amount of literature these last years. It aims to understand what rearrangement events took place and when they occurred in evolution, and it is a promising way for phylogenetic inference [1, 2].

The Genomic Median Problem (GMP) is a crucial step in genome rearrangement problems. While for only two genomes, a scenario with a minimum number of rearrangements can be reconstructed by the way of polynomial methods for

many variants of rearrangements (see for example [3, 4]), the problem is NP-hard for already three genomes [5]: it consists in searching for a fourth genome that minimizes the distance to three given genomes, in terms of the number of rearrangements.

The classical phylogenetics methods to construct an ancestral genome are based on pieces of sequences, thus making the reconstruction of the organisation of the genome impossible. One of the objectives of the GMP is to find this organisation, making a better construction of ancestral genomes. It could also be used as a hint for phylogeny, for example by using the found median as an entry-point for a phylogenetic algorithm (like in [2]).

There have been various attempts at solving the problem algorithmically. Exact solutions exist for the special case where there is only one chromosome and a rather small instance [5, 6]. Incomplete approaches, ranging from rather simple heuristics [6, 7] to more complex local search algorithms [8, 9] have been proposed. These approaches produce solutions that are often of good quality but that are not necessarily optimal and for larger instances there may be significant gaps to optimal solutions. In addition, compared to the currently available local search techniques, the approaches are rather simple and therefore one can conjecture that there is room for finding better quality solutions.

Motivated by these observations, we propose a new stochastic local search algorithm for the GMP, based on tabu search [10] and iterated local search [11]. A first goal is to improve upon the performance of current state-of-the-art algorithms in terms of run-times required to reach specific bounds on the solution quality and to find better quality solutions, thus providing new state-of-the-art solutions that may be of biological relevance. A second goal is to study the influence of the parameters on the solution process with respect to different instances; based on preliminary experiments, we have added a reactive scheme to automatically adapt crucial parameters during search; for our algorithm these are the tabu list length and the perturbation size.

The paper is organized as follows. Section 2 describes the GMP and existing approaches to solve this problem. Section 3 introduces our new stochastic local search approach, based on a combination of tabu search and iterated local search. Section 4 studies the influence of the parameters on the solution process, showing that the best parameter setting varies from one instance to another. Section 5 introduces a reactive scheme for automatically adapting parameters during search. Section 6 experimentally compares static and reactive versions of our algorithm, and also compares our algorithm with state-of-the-art approaches.

2 Problem definition and existing work

A genome will be defined as a graph, in which some edges are directed (the orthologous markers), and some not (the links between the genes).

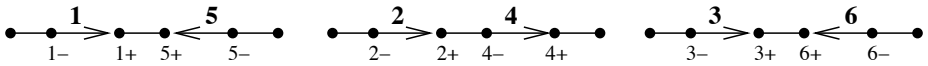
Representation of genomes by graphs. A genome is seen as a group of chromosomes. A chromosome is seen as a list of oriented genes or markers. A genome

composed of k chromosomes defined on a set of n markers is represented by a graph $G = (V, E)$.

- V associates two vertices i^- and i^+ with every marker $i \in [1; n]$ and two anonymous vertices with every chromosome $j \in [1; k]$;
- E is composed of two parts E_m and E_c:
  - for every marker i, E_m contains a *directed* edge (i^-, i^+)
  - for every chromosome c_j containing $|c_j|$ markers, E_c contains $|c_j|+1$ *non directed* edges

  The edges are defined in such a way that each chromosome corresponds to a path in G whose endpoints are anonymous vertices and which alternates directed edges —corresponding to oriented markers— and non directed edges —linking markers.

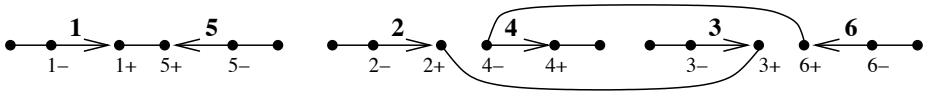
Let us consider for example a genome composed of 6 markers and the 3 following chromosomes: $c_1 = <\overrightarrow{1}, \overleftarrow{5}>$, $c_2 = <\overrightarrow{2}, \overrightarrow{4}>$, and $c_3 = <\overrightarrow{3}, \overleftarrow{6}>$. This genome is represented by the following graph.



Note that chromosomes have no directions so that paths are not directed. For example, the first chromosome c_1 is equivalent to $<\overrightarrow{5}, \overleftarrow{1}>$.

Genomic distance. A *rearrangement* in a genome is an operation that deletes two non directed edges (a, b) and (c, d), and replaces them by (a, c) and (b, d). It is the "double-cut-and-join" operation described in [4, 12], or the "2-break rearrangement" of [13]. It simulates chromosome fissions, fusions, translocations, inversions and transpositions.

Let us consider the genome of our previous example. An example of a rearrangement consists in replacing edges $(2^+, 4^-)$ and $(3^+, 6^+)$ by edges $(2^+, 3^+)$ and $(4^-, 6^+)$, thus changing chromosomes c_2 to $<\overrightarrow{2}, \overleftarrow{3}>$, and c_3 to $<\overleftarrow{4}, \overleftarrow{6}>$, as displayed in the following graph.



A rearrangement transforms one genome into another. Given two genomes G_1 and G_2 defined on the same set of markers, there is always a way to transform G_1 into G_2 by a sequence of rearrangements. The minimum number of rearrangements can be computed in linear time [4, 12]. This number is called the *genomic distance* between G_1 and G_2, and it is denoted $d(G_1, G_2)$.

The genomic distance $d(G_1, G_2)$ is computed with respect to a non directed graph $G_{1,2} = (V_{1,2}, E_{1,2})$ which is obtained by merging the two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ as follows:

- $V_{1,2} = V_1 \cup V_2$. Note that vertices associated with markers are shared by both G_1 and G_2 as they are defined on a same set of markers. However, anonymous vertices corresponding to chromosome endpoints are different in G_1 and G_2.
- $E_{1,2}$ is the union of all *non directed* edges of G_1 and G_2.

The genomic distance is defined, as in [4, 12], by $d(G_1, G_2) = n+p-c(G_{1,2})+\#(G_{1,2})/2$, where n is the number of markers, p is the number of chromosomes in both genomes, $c(G_{1,2})$ is the number of different paths and cycles in $G_{1,2}$, and $\#(G_{1,2})$ the number of paths whose endpoints are anonymous vertices that come from the same genome.

*The Genomic Median Problem (*GMP*).* Given three genomes G_1, G_2, G_3 on the same set of markers, the goal of the GMP is to determine a genome G_M that minimizes the sum $d(G_1, G_M) + d(G_2, G_M) + d(G_3, G_M)$.

The reversal median problem, which uses the number of reversal as a distance, was proven to be NP-hard [5] for the special case when the genomes are composed of one unique chromosome. The problem which we handle here uses a different distance measure, as defined in [4, 12] but most of the time the solution coincide. Also, the proof of NP-completeness is sill valid for our distance definition of the distance, which results in a problem called Cycle Median Problem in [5] and which is also proven, already for unichromosomal genomes, to be NP-complete.

Various methods for solving the *GMP* have been proposed. These comprise exact solvers or simple heuristics that work on the particular case of unichromosomal genomes like `GRAPPA` [5, 6] and some more recent heuristic improvements [7]. For the general problem case with multiple chromosomes, there is a rather simplistic search method in `MGR-MEDIAN` [8], which uses a greedy constructive algorithm. The best performance results so far have been reported for an algorithm called `MedRByLS` [9]; this is a local search algorithm inspired by `WalkSAT` [14], a well-known local search algorithm for the satisfiability problem in propositional logic. We based our algorithm on the same neighborhood and the same data structures as used in `MedRByLS`.

Neighborhood considered in MedRByLS. To find a genome G_M that minimizes the sum of the distances to three given genomes G_1, G_2 and G_3, `MedRByLS` iterately modifies a genome G_M by performing local moves. Each move corresponds to a rearrangement in G_M, that is, the exchange of two non directed edges of G_M, and is evaluated with respect to the three graphs $G_{M,1}$, $G_{M,2}$, and $G_{M,3}$.

At each step, the size of the neighborhood is in $\mathcal{O}(n^2)$, where n is the number of markers. As genomes may have several hundreds of markers, the neighborhood is reduced in [9] with respect to the following principle: a rearrangement is considered only if, in one of the three graphs $G_{M,1}$, $G_{M,2}$, and $G_{M,3}$ it breaks a cycle or a path into two cycles or paths such that one of these two cycles or paths is elementary, that is, a cycle of length 2 or a path of length 3. Therefore, moves can only increase the number of elementary cycles or paths.

Basic principle of the local search algorithm of MedRByLS. `MedRByLS` follows the random walk framework initially introduced in `WalkSAT` [14]. It starts local search from an initial genome G which may either be provided by the user or randomly chosen within the set $\{G_1, G_2, G_3\}$. Then, it iteratively chooses a move uniformly at random from the neighborhood defined above; the move is applied if it decreases the sum of the distances; otherwise it is accepted with a small probability p.

3 Tabu Search and Iterated Local Search for the GMP

The random walk framework considered by `MedRByLS` is a rather basic technique for directing the search process. Here, we consider more advanced local search approaches.

We have first re-implemented `MedRByLS`, using the same data structures based on graphs and the same neighborhood definition. This allows a direct comparison of our new algorithm to the original `MedRByLS` using a same implementation of the data structures. For comparisons that are done in Section 6, we verified that our re-implementation of `MedRByLS` matches the performance of the original version.

As a next step, we enhanced the local search by a simple tabu search scheme. For the search diversification of the resulting tabu search algorithm, we integrated it into the iterated local search framework by adding appropriate perturbations and acceptance criteria. This resulted in an algorithm that we called `MedITaS` (for Median solver by Iterated Tabu Search). More precisely, it consists of the two following main algorithmic components.

3.1 Tabu Search (TS) algorithm

First, we have implemented a simple tabu search (TS) algorithm. This algorithm forbids the reversal of the last t local search moves (where t is the tabu tenure), that is, the last changed markers. In order to do this, we use an array of n integers representing the n markers and, for each node, we put in this array the iteration when it was last changed. This simplifies the task of guessing if a move is tabu or not (that is, if it has been changed in the last cycles). Different from many other simple tabu search algorithms, ours is based on a *first-improvement* pivoting rule because the neighbourhood is very large and, thus, a full scan of it would be too time-consuming. In our experiments we have used a default initial tabu list length of 50.

3.2 Iterated Local Search (ILS) algorithm

After some preliminary tests, TS seemed to stagnate frequently in poor quality solutions. To overcome this problem, we integrated TS into an iterated local search (ILS) algorithm. ILS uses solution perturbations when the search is deemed to be stuck in plateau-moves or in a basin of attraction to generate new

starting solutions for the local search. To know if the search is stuck, we consider the resampling ratio [15], which is computed in constant time thanks to a double hash-table, and which corresponds to the percentage of solutions that are revisited with respect to the number of computed solutions. If this ratio is too high, it means that the search keeps visiting the same few solutions and it needs to escape from this search space are by a significant perturbation. We also keep track of the solution value. If this value keeps being constant for a long time, we consider being stuck in plateau-moves. In this case a perturbation could lead to avoid looking for non-interesting solutions.

The perturbation uses a rearrangement of k edges instead of 2. This means that k edges are deleted and replaced by k other edges sharing the same extremities. Finally, an acceptance criterion decides whether either the solution before the perturbation or the one after is kept for the next iteration of the ILS algorithm; in the latter case, the tabu list is emptied. The implemented acceptance criterion accepts a new solution if it is better than the previous one; otherwise, the previous solution has a user-defined probability of beeing kept (in our test, we used a default probability of 0.2, which resulted in good performance in some preliminary experimental tests).

4 Tuning of `MedITaS` parameters

In order to test the influence of parameters on the solution process, we conducted a simple experiment. First, we randomly generated 20 instances equally split on two different levels of hardness (with respect to the definition of the phase transition by [9]) but with the same size of 500 markers. The first set of 10 instances, labeled as *easy*, has a ratio of *number of markers* to *number of rearrangements* of 0.5. The second set of 10 instances, labeled as *hard*, has a value of 1.0 for the same ratio. Each of these sets has been used to off-line tune the algorithm through `F-Race` [16, 17], a tool for the automatic tuning of algorithm parameters.

For the tabu tenure of TS, the best settings resulting from these experiments were of 76 for the *easy* instances and of 86 for the *hard* instances. For the strength k of the perturbation of ILS, the best setting was 17 for the *easy* instances and it was 2 for the *hard* instances. Hence, the best settings strongly differ between the two benchmark sets, especially for the perturbation.

5 Reactive search

Experiments reported in the previous section showed that both ILS and TS are sensitive to parameter settings and that the best parameter settings are very different from one instance class to another. In addition, in further experiments we have noted that within one instance class, the best parameter settings depend further on the individual instance. Hence, we decided to extend the basic version of `MedITaS` using a reactive version of TS (adapting the tabu tenure) and ILS (adapting the perturbation strength). This reactive scheme is inspired from

reactive search [18]. It uses the resampling ratio, which is explained in Section 3.2, to determine if one of the parameters needs to be changed.

The reaction mechanism works as follows. If there is too much resampling (by default, after 3 recalculated solutions), the tabu list length is increased (by default, the size is increased by 10). At the opposite, if there is no resampling for a long time (the default value is 500 moves) the tabu list length is shortened by a parameterized number (by default, the tabu list is shortened by 1). A similar mechanism is used for tuning the ILS part. If the solution that is returned after the perturbation and the subsequent local search is an already visited solution, the reactive algorithm increases the strength k of the perturbation, because it seems that the perturbation did not succeed in escaping from the basin of attraction. Again, the size of the increase and the decrease can be parameterized; as default, we use the value 1. Finally, we should remark that the settings of the parameters that direct the reaction mechanism, at least in principle, should also be tuned. However, here we essentially stick to the default values used, assuming that, as also argued in [18], the parameters steering the reaction mechanism are more robust that the parameters tuned by the reaction mechanism.

6 Results

In order to test the efficiency of our algorithms, we ran multiple comparisons. All runs were made on the same machine having a Dual-Core AMD Opteron2216 HE (2 processors at 2.4GHz) and 4GB of RAM; only one core is used for each execution since our algorithm is implemented as a fully sequential one.

6.1 Comparison between off-line tuned and reactive algorithms

At first, we generated randomly 20 instances with a ratio of *number of markers* to *number of rearrangements* of 1.0 (which seems, according to the results from [9], to be in the phase transition) of 500 markers. This set has been split in two: 10 instances have been used to off-line tune the algorithm through `F-Race` [16, 17]; 10 other instances have been used as a test set. The algorithm to tune is the non-reactive version of `MedITaS`. We have then compared the results of the off-line tuned version to the reactive version of our `MedITaS` algorithm starting either with the default initial parameter values or with the parameter values that have been determined by the automatic tuning. For the comparison, we have run those three algorithms for 20 independent trials on each of the 10 test instances and 60 seconds per trial.

Figure 1 plots the cumulative frequency distribution of finding a bound on the median to be reached on the 10 test instances. The plot shows that, when comparing the fine-tuned version to the pure reactive version with initial default parameter values, the former gets its first high-quality solutions quicker than the latter. However, after approximately 40 seconds, the reactive version reaches higher empirical frequencies. Also, better trade-offs are obtained by initializing the reactive algorithm with the fine-tuned parameter settings: doing so gives

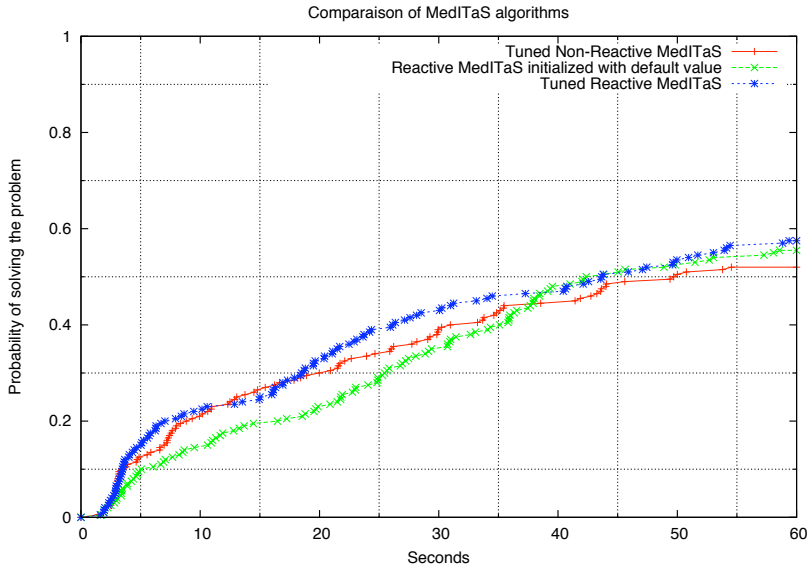


Fig. 1. Comparison between off-line tuned and reactive algorithms

good results as quickly as the non-reactive version, and, for higher computation times, it performs similar to the default reactive algorithm.

To check for the statistical significance of the results, we did pairwise comparisons of the solution quality reached using the `Wilcoxon rank sum test`, where the p-values were adjusted using the `holm` method on each of the 10 instances separately for correcting for the effect of multiple comparisons. We then counted how often the differences between two algorithms are significant (where the p-value is lower than 0.1). We run those comparison at 15, 30 and 60 seconds run-time.

From Table 1, we can clearly see that the fine-tuned reactive search performs frequently significantly better than the non-reactive version and that after 60 seconds, the non-reactive version never gets significantly better than any other method. We can also see that for a short run-time, neither of the fine-tuned reactive or non-reactive version is significantly better than the other and that the reactive method initialized with default parameters is never significantly better than any other method until 60 seconds.

	after 15 seconds			after 30 seconds			after 60 seconds		
	NR tuned	R default	R tuned	NR tuned	R default	R tuned	NR tuned	R default	R tuned
NR tuned	-	10	0	-	2	0	-	0	0
R default	0	-	0	0	-	0	1	-	1
R tuned	0	10	-	3	5	-	1	1	-

Table 1. Number of time a method x (given by the line) has been significantly better than a method y (given by the column) among the ten instances from pairwise comparisons using the `Wilcoxon rank sum test`. NR and R, respectively, stand for Non Reactive and Reactive variants of MedITas; R either starts from fine-tuned or default parameter settings whereas NR always uses fine-tuned parameter settings.

Instance	MedITas				MedRByLS				Instance	MedITas				MedRByLS			
	avg	min	max	(sdv)	avg	min	max	(sdv)		avg	min	max	(sdv)	avg	min	max	(sdv)
1	**541.0**	539	**543**	(1.0)	543.0	**538**	568	(6.9)	2	**540.4**	539	**542**	(1.0)	542.7	539	569	(6.9)
3	**563.7**	561	**566**	(1.5)	565.8	561	590	(6.9)	4	**566.9**	**565**	**568**	(0.8)	570.2	566	600	(8.6)
5	**588.9**	587	**591**	(1.3)	593.3	587	624	(8.7)	6	**593.9**	**591**	**597**	(1.7)	597.6	592	631	(9.4)
7	**609.8**	609	**613**	(1.1)	613.9	**608**	642	(8.2)	8	**610.2**	608	**612**	(1.1)	614.4	608	648	(9.3)
9	**635.1**	634	**637**	(1.0)	639.1	634	674	(9.7)	10	**637.1**	635	**639**	(1.2)	641.2	635	676	(10.1)
11	**658.5**	656	**661**	(1.4)	662.8	656	697	(10.1)	12	**653.4**	**651**	**655**	(1.1)	658.8	652	692	(9.6)
13	**669.1**	668	**672**	(1.0)	674.7	668	710	(10.6)	14	**669.3**	667	**671**	(1.2)	674.1	667	709	(10.7)
15	**675.2**	674	**677**	(1.0)	679.8	**673**	716	(10.7)	16	**684.5**	**682**	**688**	(1.6)	690.8	683	724	(10.7)
17	**693.1**	**691**	**695**	(1.1)	698.9	692	737	(11.5)	18	**692.5**	**690**	**695**	(1.4)	698.8	691	735	(11.2)
19	**707.2**	706	**709**	(0.9)	713.0	**705**	753	(11.6)	20	**705.7**	703	**709**	(1.4)	712.1	703	751	(11.6)
21	**722.3**	720	**725**	(1.1)	728.6	720	770	(12.6)	22	**716.2**	**714**	**718**	(1.0)	722.9	715	764	(12.4)

Table 2. Comparison between `MedITaS` and `MedRByLS` on randomly generated instances.

6.2 Comparison to `MedRByLS`

In another experiment, we compared the performance of our reactive `MedITaS` algorithm (starting with default parameters) to `MedRByLS`, which from a solution quality point of view is currently the state-of-the-art algorithm [9]. For this comparison, we used 22 randomly generated instances of different difficulties (with respect to the definition of the phase transition by [9]) but with the same size of 500 markers. The set has 11 levels of hardness and 2 instances per level. On this set we run our `MedITaS` algorithm and our implementation of the basic local search algorithm `MedRByLS` from [9] for 20 independent trials on each instance and 40 seconds per trial. The comparison of the best solution qualities reached by both algorithms on each instance is given in Table 2. From this table, we can see that average results computed by `MedITaS` are always better than those of `MedRByLS` (often the differences are also statistically significant) and that the gap between the two algorithms tends to increase as instances become harder. Also, the standard deviation of `MedITaS` is very low and remain constant as the instances become harder, as opposite to `MedRByLS` which has an higher deviation on harder instances. Note however that, if the worse solution found by `MedITaS` is always better than the worse solution found by `MedRByLS`, on four instances the best solution found by `MedRByLS` is better by a distance value of one than the best found by `MedITaS`.

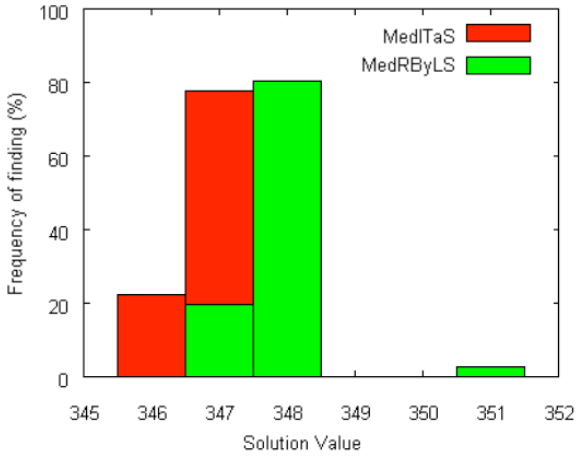


Fig. 2. Comparison between `MedITaS` and `MedRByLS` on a real-world instance (the human-mouse-rat comparison)

6.3 Real World Instance

In another experiment we used a real-world instance: the human-mouse-rat comparison, which was also used in [9]. This instance is made of 424 markers and the best median so far had a value of 346. We ran each algorithm 35 times for a computation time limit of 60 seconds. From these runs, we generated the graph in Figure 2, which represents the histogram of the frequency of finding certain solution qualities with the two main algorithms (`MedITaS`, in its reactive and fine-tuned version, and `MedRByLS`).

Figure 2 shows that `MedITaS` finds solutions that are at least as good as those found by `MedRByLS` and always of a very good quality (of 347 or better), while `MedRByLS` sometimes fails to find good ones: on some runs it returned a solution of value 351. We should also notice that `MedRByLS` has a quite low probability (less than 20%) of finding a solution of 347 or better. Finally, it should be also mentioned that in another set of experiments `MedITaS` found a new best solution for this instance with an evaluation function value of 345.

7 Discussion

Our `MedITaS` algorithm for the Genomic Median Problem gave very promising results. First, we have seen that the reactive version of our algorithm can handle relatively well a wide range of different instances without having the need to be off-line tuned. But we have also shown that a reactive search starting with fine-tuned parameters reaches a better tradeoff between solution quality and computation time than when the reactive version starts from default values. In our experimental comparison, we also have shown that `MedITaS` reaches often,

in the same computation time, better quality solutions that the `MedRByLS` algorithm, which was previously shown to be a top-performer for the same problem. Finally, it is noteworthy to be mentioned that we also found a new best solution for the human-mouse-rat common ancestor.

The developped algorithmic techniques perform very well from a solution quality point of view. However, from a biological point of view, the distance used here (as the one used in most previous attempts at solving the problem) do not seem to reflect the biological reality of the evolution process, as it is also explained in [19]). Thus, a research on a biologically more relevant operator for computing the distances has to be envisaged. We also noted in our experiments that there were a lot of medians with the exactly same value. It could be a good idea to do some comparison between them trying to extract some valuable information on the most probable characteristics of the real ancestor.

Acknowledgements.

This article is an extension of a short abstract by the same authors [20]. The authors would like to thank Yannet Interian for her kind help in any questions regarding the implementation of her algorithm. Renaud Lenne acknowledges support from the F.R.S.-FNRS through a FRIA fellowship. Thomas Stützle and Mauro Birattari acknowledge support from the Belgian F.R.S.–FNRS of which they are Research Associates. Eric Tannier is funded by the Agence Nationale pour la Recherche (ANR), projects REGLIS and GENOMICRO.

References

1. Moret, B., Tang, J., Warnow, T.: Reconstructing phylogenies from gene content and gene-order data. In Gascuel, O., ed.: Mathematics of Evolution and Phylogeny. Oxford Univ. Press (2005) 321–352
2. Bernt, M., Merkle, D., Middendorf, M.: Using median sets for inferring phylogenetic trees. Bioinformatics **23** (2007) e129–e135
3. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. Journal of the ACM **46**(1) (January 1999) 1–27
4. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics **21**(16) (2005) 3340–3346
5. Caprara, A.: The reversal median problem. INFORMS Journal on Computing **15** (2003) 93 – 113
6. Moret, B., Siepel, A., Tang, J., Liu, T.: Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In: Proceedings of the Second International Workshop on Algorithms in Bioinformatics. Volume 2452 of Lecture Notes In Computer Science., Springer-Verlag (2002) 521–536
7. Arndt, W., Tang, J.: Improving inversion median computation using commuting reversals and cycle information. In: Comparative Genomics. Volume 4751 of Lecture Notes in Computer Science., Springer Verlag (2007) 30–44

8. Bourque, G., Pevzner, P.: Genome-scale evolution: Reconstructing gene orders in the ancestral species. Genome Research **12(1)** (2002) 26–36
9. Interian, Y., Durrett, R.: Genomic midpoints: Computation and evolutionary implications (2007) Submitted.
10. Glover, F., Laguna, F.: Tabu Search. Kluwer Academic Publishers, Norwell, MA, USA (1997)
11. Stützle, T.: Iterated local search for the quadratic assignment problem. European Journal of Operational Research **174**(3) (2006) 1519–1539
12. Bergeron, A., Mixtacki, J., Stoye, J.: A unifying view of genome rearrangements. In: Proceedings of WABI 2006. Volume 4175 of Lecture Notes in Bioinformatics. (2006) 163–173
13. Alekseyev, M.A., Pevzner, P.A.: Multi-break rearrangements and chromosomal evolution. Theoretical Computer Science (2007) To appear.
14. Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the 12th National Conference on Artificial Intelligence, AAAI Press / The MIT Press, Menlo Park, CA, USA (1994) 337–343
15. van Hemert, J., Bäck, T.: Measuring the searched space to guide efficiency: The principle and evidence on constraint satisfaction. In: Proceedings of the 7th International Conference on Parallel Problem Solving from Nature. Volume 2439 of Lecture Notes in Computer Science., Springer Verlag (2002) 23–32
16. Birattari, M.: F-race: Racing methods for the selection of the best. (2005) R package version 0.1.56.
17. Birattari, M.: The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium (2004)
18. Battiti, R., Protasi, M.: Reactive local search for the maximum clique problem. Algorithmica **29**(4) (2001) 610–637
19. Eriksen, N.: Reversal and transposition medians. Theoretical Computer Science **374**(1-3) (2007)
20. Lenne, R., Solnon, C., Stützle, T., Tannier, E., Birattari, M.: Effective Stochastic Local Search Algorithms for the Genomic Median Problem. In: Doctoral Symposium on Engineering Stochastic Local Search Algorithms (SLS-DS), Brussels, Belgium (2007) 1–5