Variable Neighborhood Search for the Large Phylogeny Problem using Gene Order Data

José Luis Soncco-Álvarez
Department of Computer Science
Universidade de Brasília
Brasília, DF, 70910-900, Brazil
Email: jose.soncco.alvarez@gmail.com

Mauricio Ayala-Rincón
Departments of Mathematics and Computer Science
Universidade de Brasília
Brasília, DF, 70910-900, Brazil
Email: ayala@unb.br

***Abstract*—Computing evolutionary distances using gene order data is a complex combinatory problem; nevertheless, for specific metrics exact polynomial algorithms were proposed, having in many cases non trivial approaches. This scenario can become harder if we want to reconstruct phylogenies based on gene order data: first it is necessary to explore the search space of possible tree structures which is well-known to be exponential; second, it is necessary a method for evaluating the cost of these trees, i.e. to find a labeling of the internal nodes that leads to the most parsimonious cost of a tree under a given evolutionary distance. The latter problem was shown to be $\mathcal{NP}$-hard even for 3 genomes (median problem) under many evolutionary distances. In this paper we propose a variable neighborhood search approach for solving the large phylogeny problem for data based on gene orders. Also, a greedy approach is proposed for the small phylogeny problem aiming to reduce the running time of the Kovac et al. dynamic programming approach. Our proposed algorithms were implemented as the software called `HELPHY`. Experiments showed that the running time is improved for finding trees with good scores (reversal distance) for the *Campanulaceae* dataset, and a new tree structure was found having the best known score (double cut and join distance) for the case of *Hemiascomycetes* dataset.**

I. Introduction

Reconstructing phylogenies for organisms that are represented as gene orders brings challenges such as calculating the evolutionary distance between two genomes, or calculating the median of three genomes. This problem can be divided into an outer and an inner problem. The outer problem is known as the *large phylogeny problem* and aims to explore the search space of trees and found those with the best parsimonious values. Blanchette et al. [1] implemented a software called `BPAnalysis` that explores all the search space, this method is unfeasible when using large inputs. Moret et al. [2] addressed this problem with the software `GRAPPA` by sampling the search space (and also using a parallel approach) making it possible to use larger inputs. The inner problem is known as the *small phylogeny problem* and aims to label the internal nodes of a given tree structure such that the cost of the tree is the most parsimonious possible. Most of the solutions to this problem are based on solving the median problem defined over an evolutionary distance [1], [2], [3], [4], [5], [6].

In this paper we propose two new algorithms: a *variable neighborhood search* (VNS) approach to explore the search space and solve the large phylogeny problem, and a greedy approach to label the internal nodes of a tree and solve the small phylogeny problem, in both cases we deal with gene order data. The algorithms were implemented as the system called `HELPHY` - **He**uristics for the **L**arge (and Small) **Phy**logeny Problem. Experiments were performed using two important datasets of the literature: the *Campanulaceae* and the *Hemiascomycetes* datasets. `HELPHY` found, in just 1.27 minutes (76.2 seconds), many trees with scores 64 and 65 reversals for the large phylogeny problem using the *Campanulaceae* dataset, these are very competitive scores since the best trees found in the literature have score of 64 reversals. Also, regarding the *Hemiascomycetes* dataset, `HELPHY` found many trees with scores 76, 77 and 78 *double cut and join* (DCJ) operations. Further, `HELPHY` improved the score of one of these trees to 73 DCJ operations by applying the approach to deal with the small phylogeny problem. This tree structure is different from the one used in the literature [7], [8] and the score is the best found as far as we know for the *Hemiascomycetes* dataset.

Section II presents some definitions and related work; Section III, the greedy algorithm for the large phylogeny problem; Section IV, the variable neighborhood search approach for the large phylogeny problem. Then, Section V shows the experiment settings and the results. Finally, Section VI concludes and suggests future work.

II. Background

A. Definitions

Some of the definitions of this section were taken mainly from [9], [6], [10], [3], [4].

A **gene** is represented as a signed integer, where the integer represents the gene order inside a genome, and the sign indicates its orientation. A **genome** is a set of chromosomes, and each chromosome is a sequence of genes.

Let G be a genome with a single chromosome with genes $g_1,\ g_2,\ \ldots, g_i,\ g_{i+1},\ \ldots,\ g_{j-1},\ g_j,\ \ldots,\ g_n$, a **reversal** of the subsequence between the i^{th} and the j^{th} genes ($i \leq j$) is an operation that transforms G into $g_1,\ g_2,\ \ldots,\ -g_j,\ -g_{j-1},\ \ldots,\ -g_{i+1},\ -g_i,\ \ldots,\ g_n$.

The **reversal distance** is the minimum number of reversals necessary to transform a genome into another.

The orientation of genes might also be expressed by ordering the head and tail of each gene. Let a_h and a_t be the head and tail of a gene a respectively, which are known as **extremities** of the gene. In case gene a has a positive sign the extremities have the order a_t and a_h, otherwise the gene extremes have the order a_h and a_t.

An **adjacency** of two consecutive genes a and b can have one of the following forms: $\{a_h, b_h\}, \{a_h, b_t\}, \{a_t, b_h\}, \{a_t, b_t\}$. A **telomere** is an extremity of a gene a that is not adjacent to any other gene and is represented as $\{a_h\}$ or $\{a_t\}$.

A **double cut and join** operation acts on two points (adjacencies or telomeres) u and v of a genome and is defined as follows:

(a) If $u = \{p, q\}$ and $v = \{r, s\}$ are adjacencies, these are replaced either by the adjacencies ($\{p, r\}$ and $\{s, q\}$) or ($\{p, s\}$ and $\{q, r\}$).
(b) If $u = \{p, q\}$ is an adjacency and $v = \{r\}$ is an telomere, these are replaced either by ($\{p, r\}$ and $\{q\}$) or ($\{q, r\}$ and $\{p\}$).
(c) If both $u = \{q\}$ and $v = \{r\}$ are telomeres, these are replaced by $\{q, r\}$.
(d) If $u = \{p, q\}$ is an adjacency, then it is replaced by the telomeres $\{p\}$ and $\{q\}$.

The **DCJ distance** is the minimum number of DCJ operations necessary to transform a genome into another.

A **phylogenetic tree** for a set S of n genomes is an unrooted binary tree with n leaves, so that every leaf of the tree is labeled with a distinct element of S.

Given a phylogenetic tree and a labeling of its internal nodes, the **cost of the tree** is the sum of the costs of all its edges, where the cost of an edge is the evolutionary distance between the genomes labeling the extremes of the edge.

Given a set S of n genomes and a specific evolutionary distance (such as reversal or DCJ), the **large phylogeny problem** is to find a tree and a labeling of its internal nodes such that the cost of the tree is minimum. The number of all possible unrooted binary trees for n genomes is:

$$\prod_{i=3}^{n}(2i-5) = \frac{(2n-4)!}{2^{n-2}.(n-2)!}$$

This exponential search space makes this problem challenging.

Given a set S of n genomes, a specific evolutionary distance, and a structure of a tree, the **small phylogeny problem** is to find a labeling for the internal nodes of the tree such that the cost of the tree is minimum.

The minimum case ($n = 3$) of the small phylogeny problem is known as the **median problem** for three genomes, which was shown to be $\mathcal{NP}$-Hard for various evolutionary models [11], [12], [13], [14]. Fig. 2 shows the DCJ problem for 3 genomes A,B, and C, where M is the median genome to be found. The cost of this tree, to be minimized, is given by the sum of $d_1 + d_2 + d_3$.

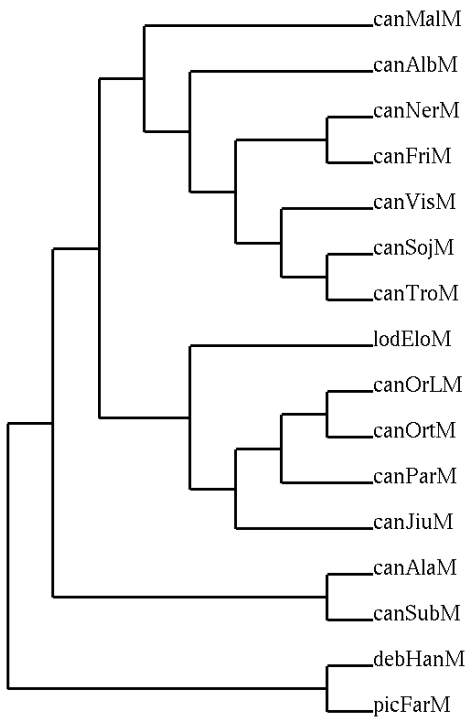

Fig. 1. *Hemiascomycetes* Tree found by using Software `MrBayes` in [7].

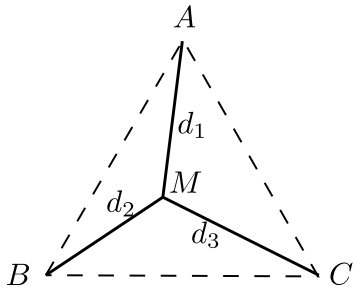

Fig. 2. Median Problem for Three Genomes A, B, and C.

B. Related Work

One of the most used methods for solving the small phylogeny problem using gene order data is the method presented by Blanchette et al. [15], [1] (adapted from the iterative method proposed at [16]) in which the score of a given tree is optimized (minimized) by solving iteratively the median of three neighbors until no more improvement can be done. In these works, it is used a general distance between two genomes A and B called breakpoint distance, so that two adjacent genes in A define a breakpoint if these genes are not adjacent in B, these definition can easily be adapted to signed genes [15]. In this approach, the median problem for breakpoints is reduced to an instance of the traveling salesman problem (TSP) and then solved using a TSP exact solver. This approach was implemented in a software called `BPAnalysis` to solve the large phylogeny problem by exploring the entire search space, which makes this method extremely slow for larger datasets

such as the *Campanulaceae* as reported in [2].

In order to overcome the shortcoming of BPAnalysis, Moret et al. [2] reimplemented it in a software called GRAPPA. This software explores a sample of the search space, and also uses a technique of condensation of genomes to reduce the length of the genomes. Moreover, greedy and exact TSP solvers (for the median problem for *breakpoints*) were implemented and reported to be faster than the ones used by BPAnalysis.

The linear time algorithm for calculating the reversal distance [17] was included in the analysis performed by GRAPPA in [18]. Experiments for the *Campanulaceae* dataset returned 216 trees with a score of 67 reversals. Later, Moret et al. [19] included in GRAPPA the algorithm to solve the reversal median problem proposed by Caprara [20]. Experiments with the *Campanulaceae* dataset returned many trees with a score of 64 reversals in just one hour running on a single workstation [19].

Bourque and Pevzner [3] proposed a different approach for solving the reversal median problem of three genomes (say G_1, G_2, G_3). In this approach, the heuristic of applying *good reversals* over a genome G_1 is applied. According to this heuristic, a reversal is selected whenever it leads G_1 to be closer to the other two genomes G_2 and G_3. This heuristic is applied also to G_2 and G_3 until converging to a common ancestor. For building a tree, a new genome is added iteratively into some edge of a partial tree. This edge is the one that forms an instance of the median problem with the new genome and that lead to a small score. This algorithm was implemented into a software called MGR. Experiments with the *Campanulaceae* datasets returned a tree with a score of 65 reversals.

Larget et al. [21] proposed a bayesian approach which was implemented in a software called BADGER and found 180 different trees with a score of 64 reversals for the *Campanulaceae* dataset.

Adam and Sankoff [4] implemented an algorithm for the small phylogeny problem that iteratively solves the median problem, but using as metric the DCJ distance (proposed by Yancopoulos et al. [22] and restated by Bergeron et al. [9]). The heuristic used to solve the DCJ median problem is similar to the one implemented in MGR, that is, for 3 genomes that forms an instance of the median problem, a DCJ operation is chosen whenever it makes one genome closer the other two genomes. Experiments with the *Campanulaceae* dataset and the tree found by MGR [3] returned a score of 59 DCJ operations, but when internal nodes were restricted to not have additional circular chromosomes the score of the tree was 64 DCJ operations.

Up to this point, the median problem showed to be the most important problem to be tackled in order to solve the small phylogeny problem. In this sense efforts were directed to solve the DCJ median problem. Xu and Sankoff [5] proposed and exact branch and bound algorithm to solve the DCJ median problem which was implemented in a software called ASMedian. Later, Gao et al. [6] proposed a genetic algorithm for the DCJ median problem. Experiments showed that Gao et al. approach gave competitive results regarding ASMedian, and improved the running time when harder instances of the problem were used.

Kováč et al. [7] proposed a different approach (implemented in a software called PIVO) for solving the small phylogeny problem that consists in the generation of candidates for each internal node. Each candidate is generated by applying a DCJ operation over the genome labeling an internal node. The next step is to score the candidates by a dynamic programming algorithm, then the candidates with the best scores are chosen to label the internal nodes. This process is repeated until the overall score of the tree can not be improved more. Experiments with *Campanulaceae* dataset and the tree structure found by MGR [3], returned a score of 59 DCJ operations, and a score of 64 DCJ operations for the restricted case (just one circular chromosome is allowed for the internal nodes). An additional experiment was performed using the *Hemiascomycetes* dataset for which the tree structure (see Fig. 1) used was calculated by using the software MrBayes [23], this experiment returned a score of 78 restricted DCJ operations (either one circular chromosome, or one or more linear chromosomes are allowed for the internal nodes).

Afterwards, Herencsár and Brejová [8] proposed many heuristic such as the tabu search for improving the PIVO software, and implemented it in the software PIVO2. Experiments showed that PIVO2 returned better scores regarding PIVO for the *Campanulaceae* and *Hemiascomycetes* datasets, these result are summarized in the Table I.

III. Greedy Algorithm for the Small Phylogeny Problem

Based on the approach proposed by Kováč et al. [7] (we will call it **Kovac-Opt**) we propose a greedy approach (instead of using dynamic programming) for solving the small phylogeny problem. The main reason for proposing a greedy approach (we wil call it **Greedy-Opt**) was to decrease the running time used by PIVO, which was extremely slow.

Algorithm 1 shows the pseudocode of the greedy approach, the main idea is to generate a set of candidates for a given node i that are far by one DCJ operation. The candidate that has the minimum score is chosen to replace the node i. The score of a candidate is calculated by adding the distances of the candidate regarding the i ancestor, i left ancestor, and i right ancestor.

The main drawback of generating candidates is that number of candidates can be huge, thus increasing the processing time. In order to overcome this drawback the set of candidates was reduced by avoiding the application of DCJ operations over adjacencies that also appear in the i ancestor, i left ancestor, and i right ancestor at the same time.

An important improvement taken from PIVO [7] is to reuse as candidates the labeled internal nodes that were found by Algorithm 1 in a previous run. This implies that we should run some iterations of Algorithm 1 in order to exploit this improvement.

Algorithm 1: Greedy Algorithm for the Small Phylogeny Problem

Input: Dataset of genomes, Tree structure
Output: Score of the tree

Perform an initial labeling of internal nodes;
$newScore \leftarrow$ Calculate cost of the tree;
$score \leftarrow \infty$;
while $newScore < score$ **do**
  $score \leftarrow newScore$;
  foreach *internal node* i **do**
    $d_1, d_2, d_3 \leftarrow$ Calc. distances of i and its ancestor, i and its left desc., and i and its right desc. respectively;
    $d \leftarrow d_1 + d_2 + d_3$;
    Generate candidates for node i;
    foreach *candidate* c **do**
      $nd_1, nd_2, nd_3 \leftarrow$ Calc. distances of c and i ancestor, c and i left desc., and c and i right desc. respectively;
      $nd \leftarrow nd_1 + nd_2 + nd_3$;
      if $nd < d$ **then**
        $d \leftarrow nd$;
        Save candidate c;
      end
    end
    Replace internal node i by saved candidate c if it exists;
  end
  $newScore \leftarrow$ Calculate cost of the tree;
end
Recover last state of labeled nodes;
Return $score$;

IV. VNS for the Large Phylogeny Problem

The search space of tree structures related to the large phylogeny problem when using gene order data was explored either exhaustively [15], [1] or by sampling [2], [18]. As far as we know, heuristics for exploring this search space in the context of gene order data were not applied, despite for the case of using sequence of characters (such as DNA sequences) there is extensive literature for reconstructing phylogenetic trees using heuristics and evolutionary approaches [24], [25], [26], [27], [28].

In the current work we are proposing a VNS approach for the large phylogeny problem using gene order data. VNS is a meta-heuristic proposed by Mladenović and Hansen [29], this framework changes among neighborhood structures searching for an optimal (or near-optimal) solution [30].

Given a solution T' (tree structure) we say that it is a neighbor of T (tree structure), if the former can be reached from the latter in a single step (by applying a move operator). The neighborhood structure $\mathcal{N}(T)$ of a solution T is the set of all its neighbors [31].

For the case of reconstructing phylogenetic trees using as input data sequence of characters, Andreatta and Ribeiro [25] used the VNS variant known as General VNS with three neighborhood structures. In our work, a simpler VNS variant is used known as Reduced VNS (see [30] for this variant) with four neighborhoods structures. Algorithm 1 is used for evaluating the score (cost) of a given tree structure.

Algorithm 2 shows the Reduced VNS for the large phylogeny problem, this algorithm was implemented in the system `HELPHY`. In this algorithm, four neighborhood structures are used and are represented as $\mathcal{N}_k,\ k = 1, \ldots, k_{max} = 4$. These structures will be presented in the next subsections.

Algorithm 2: (Reduced) VNS for the Large Phylogeny Problem

Input: Dataset of genomes
Output: A tree structure with a score

Generate an initial solution T (tree structure);
$score \leftarrow$ Calculate cost of T (Algorithm 1);
$iteration \leftarrow 1$;
while $iteration \leq maxIterations$ **do**
  $k \leftarrow 1$;
  while $k \leq k_{max}$ **do**
    Shaking:
    Generate randomly a solution T' (tree structure) for the neighborhood structure $\mathcal{N}_k(T)$;
    $newScore \leftarrow$ Calculate cost of T' (Algorithm 1);
    Move or not:
    if $newScore < score$ **then**
      $score \leftarrow newScore$;
      $T \leftarrow T'$;
      $k \leftarrow 1$;
    end
    else
      $k \leftarrow k + 1$;
    end
  end
end
Return tree T and $score$;

As a further step, a refinement procedure is executed for the tree found by Algorithm 2, this procedure consists in finding the best neighbors for the following neighborhoods structures: $\mathcal{N}_5$ and $\mathcal{N}_6$ (see next subsections).

Instead of defining a neighborhood structure by enumerating all of its neighbors, we can define it implicitly by referring to the potential transitions that can be reached after the application of a move operator [31]. In the following subsections it will be presented the procedure used to generate the initial solution and the neighborhood structures already mentioned in this section: $\mathcal{N}_1, \ldots, \mathcal{N}_6$.

A. Generating Initial Solutions

Given a set S of genomes (leaves), we can construct an initial phylogenetic tree T by following the next steps:

1) Remove a random element g from S.
2) Insert g into some edge of tree T.
3) Go to step 1) or stop if the set S is empty.

There are many policies for inserting a genome into a tree, which give rise to variants of this basic algorithm [25]. The variant used in this work is the one that inserts a genome into the edge that leads to the first minimum increase in the cost of the tree. Andreatta and Ribeiro [25] showed that this variant is the one with the best trade-off between accuracy and running time.

B. Neighborhood Structures

The neighborhood structures that were used in our work are the following:

- $\mathcal{N}_1$: Nearest Neighborhood Interchange (NNI) [32], [25]. A neighbor is generated by following the next steps: (1) select randomly (from a tree) an internal edge, which will have four subtrees connected to it; (2) swap the position of two non adjacent subtrees, i.e., subtres not sharing the same internal node.
- $\mathcal{N}_2$: Single Step (STEP) [25]. A neighbor is generated by taking out a leaf node from a tree and inserting it into another edge.
- $\mathcal{N}_3$: Subtree Pruning and Regrafting (SPR) [25], [33], [34], [35]. A neighbor is generated by following the next steps: (1) take out an internal node (from a tree), giving rise to three subtrees; (2) join two subtrees by an edge; (3) join the remaining subtree by an edge different from the original position.
- $\mathcal{N}_4$: Tree Bisection and Reconnection (TBR) [36], [35]. A neighbor is generated by following the next steps: (1) take out an edge (from a tree), giving rise to two subtrees; (2) reconnect the two subtrees by any pair of edges of the first and second subtrees;
- $\mathcal{N}_5$: Subtree Scramble (SCRAMBLE) [37]. A neighbor is generated by selecting randomly a subtree (from a tree) and then rearranging randomly its structure.
- $\mathcal{N}_6$: Leaf Swap (SWAP) [37]. A neighbor is generated by selecting randomly two leaves (from a tree) and then swapping their positions.

V. Experiments

Experiments are presented, which were performed for the small and large phylogeny problem and using as input two datasets of the literature: the *Campanulaceae* cpDNA [10], [38] and the *Hemiascomycetes* mtDNA [39] datasets.

Experiments were executed in an `OSX` operating system using a i7 processor with 16 GB of RAM. Also, the `HELPHY` software was compiled with `gcc` using the `-O2` option.

Note that the restricted DCJ distance have different definitions for each dataset. In the case of the *Campanulaceae* dataset, the restricted DJC considers only ancestor genomes (internal nodes) with one circular chromosome. In the case of the *Hemiascomycetes* dataset, the restricted DCJ considers either ancestor genomes (internal nodes) with one circular chromosome, or with one or more linear chromosomes.

A. Experiments for the Small Phylogeny Problem

Experiment 1. For each dataset and each distance (DCJ and Restricted DCJ) the following experiment was performed in order to find the best possible scores.

- Execute `HELPHY` 50 times for the small phylogeny problem, using the re-implementation of Kovac-Opt, and save the best score found. The Kovack-Opt has 10 iterations.
- Execute `HELPHY` 50 times for the small phylogeny problem, using the Greedy-Opt (Algorithm 1), and save the best score found. The Greedy-Opt has 10 iterations.

Table I shows the results (tree scores), in the two last lines, using the configuration of Experiment 1 for the *Campanulaceae* dataset. The tree structure was taken from [3] with a reversal score of 65. Note that `HELPHY`, using the re-implementation Kovac-Opt, achieved the same results as `PIVO` (for DCJ and Restricted DCJ distance). Thus for the *Campanulaceae* dataset the replication of results was successful even when `PIVO` represents the phylogenies as rooted binary trees. Also, `HELPHY` with the Greedy-Opt, achieved the same results as `PIVO` for the DCJ distance, but not for the Restricted DCJ distance.

TABLE I
Tree Scores Found by `HELPHY` for the Small Phylogeny Problem using the *Campanulaceae* Dataset

	Reversal Distance	DCJ	Restricted DCJ
`GRAPPA` [18]	67	-	-
`MGR` [3]	65	-	-
`GRAPPA` [19]	64	-	-
`BADGER` [21]	64	-	-
`ABC` [4]	-	59	64
`PIVO` [7]	-	59	62
`PIVO2` [8]	-	56	59
`HELPHY` (Kovac-Opt [1])	-	59	62
`HELPHY` (Greedy-Opt)	-	59	63

Table II shows the time comparison for the two tree optimizers (Kovac-Opt and Greedy-Opt) used by `HELPHY` after 50 executions, using the *Campanulaceae* dataset and the DCJ distance. From this table, it can be observed that `HELPHY` with the Greedy-Opt is 6.72 times faster than `HELPHY` using the Kovac-Opt.

TABLE II
Time Comparison of the Tree Optimizers used by `HELPHY` for the Small Phylogeny Problem using the *Campanulaceae* Dataset (for DCJ)

	Total Time	Avg. Time
`HELPHY` (Kovac-Opt)	4.64 min	5.57 sec
`HELPHY` (Greedy-Opt)	0.69 min	0.83 sec

Table III shows the results (tree scores), in the two last lines, using the configuration of the Experiment 01 for the *Hemiascomycetes* dataset. The tree structure used as input was calculated in [7] by using the program `MrBayes` [23].

[1]This is the re-implementation of Kovac-Opt proposed in [7]

From this table it can be observed that HELPHY, using a re-implementation of Kovac-Opt, has lower score than PIVO2 just regarding the DCJ distance.

TABLE III
TREE SCORES FOUND BY HELPHY FOR THE SMALL PHYLOGENY PROBLEM USING THE *Hemiascomycetes* DATASET

	DCJ	**Restricted DCJ**
PIVO [7]	-	78
PIVO2 [8]	75	77
HELPHY (Kovac-Opt)	74	79
HELPHY (Greedy-Opt)	77	82

Table IV shows the time comparison for the two optimizers used by HELPHY after 50 executions, using the *Hemiascomycetes* dataset and the DCJ distance. From this table it can be observed that HELPHY with Greedy-Opt is 3.95 times faster than HELPHY with Kovac-Opt.

TABLE IV
TIME COMPARISON OF THE TREE OPTIMIZERS USED BY HELPHY FOR THE SMALL PHYLOGENY PROBLEM USING *Hemiascomycetes* DATASET (FOR DCJ)

	Total Time	**Avg. Time**
HELPHY (Kovac-Opt)	2.49 min	2.99 sec
HELPHY (Greedy-Opt)	0.63 min	0.77 sec

B. Experiments for the Large Phylogeny Problem

The results of the previous experiment gave us valuable information for choosing the tree optimizer that is going to be used as default by HELPHY for solving either the large or small phylogeny problem. Indeed, Greedy-Opt is faster than Kovac-Opt and then is the default option when dealing with the large phylogeny problem, due to the exponential tree space to be explored. However, when dealing with the small phylogeny problem the main aim is to reduce the score for a fixed tree structure, and in this case the default option would be the Kovac-Opt. Both optimizers have as default ten iterations for the small phylogeny problem, and one iteration for the large one.

Experiment 2. In order to explore the tree structure space and find a tree structure with the best possible score the following experiment was performed for each dataset:

- Execute HELPHY 50 times for the large phylogeny problem and save the tree structures with their scores (reversal or DCJ).
- Choose those tree structures with the best scores.

Experiment 3. After finding a set of tree structures with good scores the following experiment is performed for each distance (DJC and Restricted DCJ) and each tree in order to reduce their scores:

- Execute HELPHY 50 times for the small phylogeny problem and save the best score found (DJC or Restricted DCJ).

Table V shows the results of experiments 2 and 3 for the *Campanulaceae* dataset. In the second column it appears the score of the best trees found (with 64 and 65 reversals) for the large phylogeny problem (Experiment 2). The third and fourth columns show the scores (DCJ and Restricted DCJ) found by HELPHY for the Experiment 3. Note that two trees (Tree32 and Tree33) were found having the best scores: 64 reversals, 59 DCJ, and 62 Restricted DCJ. These results are equal to those found by the PIVO algorithm using the tree structure found by MGR [3].

TABLE V
TREES (AND SCORES) FOUND BY HELPHY FOR THE LARGE (AND SMALL) PHYLOGENY PROBLEM USING THE *Campanulaceae* DATASET

	Large Phylogeny (Reversal)	**Small Phylogeny (DCJ)**	**Small Phylogeny (Restricted DCJ)**
Tree32	**64**	**59**	**62**
Tree33	**64**	**59**	**62**
Tree40	**64**	**59**	63
Tree1	65	**59**	**62**
Tree11	65	61	63
Tree12	65	60	65
Tree18	65	61	**62**
Tree26	65	61	63
Tree27	65	61	63
Tree28	65	61	63
Tree38	65	61	63
Tree6	65	61	63

Table VI shows the running time of HELPHY for the experiments performed to generate Table V. HELPHY took just 1.27 minutes to find three trees with a score of 64 reversals and nine trees with a score of 65 reversals. When dealing with the reversal distance HELPHY uses the algorithm that iteratively solves instances of the median problem to score a tree. The routine for solving the reversal median problem was taken from GRAPPA. For the small phylogeny problem HELPHY took 66.04 minutes and 70.52 minutes for the DCJ and Restricted DCJ distance, respectively. The overall time spent for analyzing the *Campanulaceae* dataset is 137.83 minutes (2.29 hours).

TABLE VI
RUNNING TIME OF HELPHY FOR THE LARGE AND SMALL PHYLOGENY PROBLEMS USING THE *Campanulaceae* DATASET

	Time (min)
Large Phylogeny (Reversals)	1.27
Small Phylogeny (DCJ)	66.04
Small Phylogeny (Restricted DCJ)	70.52
Total	137.83

Table VII shows the results of experiment 2 and 3 for the *Hemiascomycetes* dataset. In the second column it can be observed the score of the best trees found (76, 77, and 78 DCJ) for the large phylogeny problem. The third and fourth columns show the scores (DCJ and Restricted DCJ) found by HELPHY for the small phylogeny problem (Experiment 3). It was found a tree (Tree4) with the following best score: 73 DCJ and 76 Restricted DCJ. This tree has better scores than

the ones found by PIVO2 (and PIVO) for the tree structure found using MrBayes.

TABLE VII
TREES (AND SCORES) FOUND BY HELPHY FOR THE LARGE (AND SMALL) PHYLOGENY PROBLEM USING THE *Hemiascomycetes* DATASET

	Large Phylogeny (DCJ)	**Small Phylogeny (DCJ)**	**Small Phylogeny (Restricted DCJ)**
Tree48	**76**	76	82
Tree5	77	76	84
Tree20	77	75	77
Tree30	77	**73**	80
Tree4	78	**73**	**76**
Tree11	78	76	79
Tree22	78	74	80
Tree42	78	75	78
Tree44	78	76	79

Table VIII shows the running time of HELPHY for the experiments performed to generate Table VII. HELPHY took 38.37 minutes to find one tree with a score of 76 DCJ, three trees with a score of 77 DCJ, and five trees with a score of 78. For the case of the small phylogeny problem, HELPHY took 19.34 minutes and 20.73 minutes for the DCJ and Restricted DCJ distance, respectively. The overall time spent for analyzing the *Hemiascomycetes* dataset is 78.44 minutes (1.31 hours).

TABLE VIII
RUNNING TIME OF HELPHY FOR THE LARGE AND SMALL PHYLOGENY PROBLEMS USING THE *Hemiascomycetes* DATASET

	Time (min)
Large Phylogeny (DCJ)	38.37
Small Phylogeny (DCJ)	19.34
Small Phylogeny (Restricted DCJ)	20.73
Total	78.44

VI. CONCLUSION

In this paper we proposed a greedy approach for the small phylogeny problem and a variable neighborhood search approach for the large phylogeny problem, both for dealing with gene order data. Algorithms were implemented in the HELPHY system.

Two important cases from the literature were used: the *Campanulaceae* and the *Hemiascomycetes* datasets. Results showed that the greedy approach improves the running time regarding the dynamic programming approach, but loses accuracy. Based on these results the greedy approach was used as default (case of DCJ distance) for evaluating the cost of the trees for the large phylogeny problem and, the dynamic programming approach proposed by Kováč et al. [7] was used as default (case of DCJ distance) for the small phylogeny problem. For the case of the reversal distance, Caprara's algorithm for the reversal median problem (taken from GRAPPA) was used to evaluate the cost of a tree structure by using the iterative algorithm presented by Blanchette et al. [15].

The HELPHY software showed to be suitable for analyzing datasets using the reversal and DCJ distances. In the case of the *Campanulaceae* dataset the running time was improved to just 1.27 minutes for discovering three trees of score 64 reversals and nine trees of score 65 reversals, since the best running time found in the literature [19] was of one hour for finding trees of score 64 reversals. For the case of the *Hemiascomycetes* dataset, a modest running time of 38.37 minutes was used for finding one tree of score 76 DCJ, three trees of score 77 DCJ, and five trees of score 78 DCJ. However, for reducing these scores it was necessary to run HELPHY for the small phylogeny problem, increasing the overall running time to 1.31 hours, but improving the score of the trees, with one of them having the score of 73 DCJ (and 76 restricted DCJ). This tree structure found (see Fig. 3) for the *Hemiascomycetes* dataset is different (and improves the score) from the one found by MrBayes program in [8] with score of 75 DCJ (and 77 restricted DCJ).

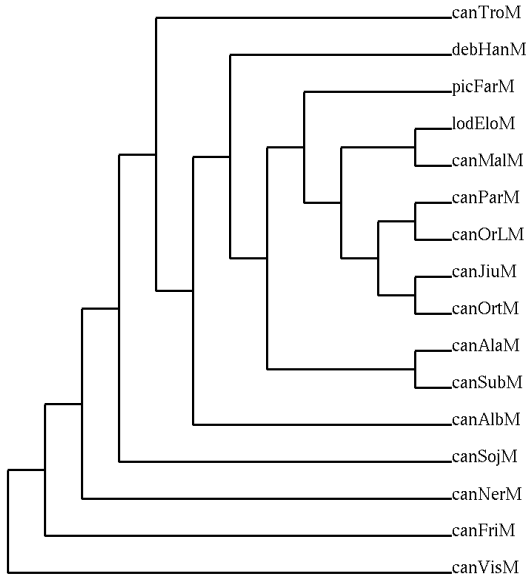

Fig. 3. *Hemiascomycetes* Tree found by HELPHY.

As a future work, we are planning to implement other meta-heuristics algorithms such genetic algorithms, memetic algorithms, and simulated annealing. For these evolutionary approaches the Algorithm 1 could be used as fitness function. Also, we will test parallel approaches for the variable neighborhood search such the ones commented in [30], this will reduce the overall running time for analyzing datasets for the large phylogeny problem. Finally, it is interesting to test the HELPHY software with other datasets based on gene orders.

APPENDIX

Best trees in newick format from Table V:

- **Tree32**: (Platycodon,((Tobacco,(((Legousia,Triodanus), Asyneuma),(((Campanula,Adenophora),Trachelium), ((Merciera,Wahlenbergia),Symphyandra)))), (Codonopsis,Cyananthus)))
- **Tree33**: (Adenophora,(Campanula,(((((Codonopsis, Cyananthus),(Tobacco,Platycodon)),(Asyneuma, (Triodanus,Legousia))),((Merciera,Wahlenbergia), Trachelium)),Symphyandra)))

- **Tree40**: (Merciera,(Wahlenbergia,((((((Codonopsis, Cyananthus),Platycodon),Tobacco),((Triodanus,Legousia) ,Asyneuma)),(Symphyandra,(Adenophora,Campanula))), Trachelium)))
- **Tree1**: (Triodanus,(Legousia,(((Symphyandra, ((Campanula,Adenophora),((Wahlenbergia,Merciera), Trachelium))),((Platycodon,Tobacco),(Cyananthus, Codonopsis))),Asyneuma)))

Best trees in newick format from Table VII:

- **Tree48**: (canJiuM,((canAlaM,((canSubM,(((canAlbM,debHanM), picFarM),(canMalM,lodEloM))),((canNerM,canFriM), ((canSojM,canTroM),canVisM)))),(canOrtM, (canOrLM,canParM))))
- **Tree5**: (canAlaM,((canSojM,(((canFriM,canVisM),(canAlbM, canTroM)),canNerM)),((((lodEloM,canMalM),picFarM) ,canSubM),(debHanM,(canJiuM,(canOrtM, (canParM,canOrLM)))))))
- **Tree20**: (canSojM,((canFriM,((canTroM,canNerM),canVisM)), (canAlbM,(canMalM,(((lodEloM,(picFarM,debHanM)), canAlaM),(canSubM,(canJiuM,(canOrtM, (canOrLM,canParM)))))))))
- **Tree30**: (lodEloM,((canAlaM,canSubM),((((canMalM,debHanM), (canJiuM,(canOrtM,(canOrLM,canParM)))),picFarM), ((canAlbM,(canVisM,((canNerM,canFriM),canTroM))), canSojM))))
- **Tree4**: (canVisM,(canFriM,(canNerM,(canSojM,((canAlbM, (((canSubM,canAlaM),((((canOrtM,canJiuM), (canOrLM,canParM)),(canMalM,lodEloM)),picFarM)), debHanM)),canTroM)))))

REFERENCES

[1] M. Blanchette, T. Kunisawa, and D. Sankoff, "Gene order breakpoint evidence in animal mitochondrial phylogeny," *Journal of Molecular Evolution*, vol. 49, no. 2, pp. 193–203, 1999.

[2] B. Moret, S. Wyman, D. Bader, T. Warnow, and M. Yan, "A new implementation and detailed study of breakpoint analysis," in *Proc. 6th Pacific Symp. on Biocomputing (PSB01)*, no. LCBB-CONF-2001-007. World Scientific Pub., 2001, pp. 583–594.

[3] G. Bourque and P. A. Pevzner, "Genome-scale evolution: reconstructing gene orders in the ancestral species," *Genome research*, vol. 12, no. 1, pp. 26–36, 2002.

[4] Z. Adam and D. Sankoff, "The abcs of mgr with dcj," *Evolutionary Bioinformatics*, vol. 4, 2008.

[5] A. W. Xu and D. Sankoff, "Decompositions of multiple breakpoint graphs and rapid exact solutions to the median problem," in *Algorithms in Bioinformatics*. Springer, 2008, pp. 25–37.

[6] N. Gao, N. Yang, and J. Tang, "Ancestral genome inference using a genetic algorithm approach," *PloS one*, vol. 8, no. 5, p. e62156, 2013.

[7] J. Kováč, B. Brejová, and T. Vinař, "A practical algorithm for ancestral rearrangement reconstruction," in *International Workshop on Algorithms in Bioinformatics*. Springer, 2011, pp. 163–174.

[8] A. Herencsár and B. Brejová, " An Improved Algorithm for Ancestral Gene Order Reconstruction," in *Information Technologies - Applications and Theory (ITAT)*, ser. CEUR-WS, V. Kurkova, L. Bajer, and V. Svatek, Eds., no. 1003, Jasna, Slovakia, 2014, pp. 46–53.

[9] A. Bergeron, J. Mixtacki, and J. Stoye, "A unifying view of genome rearrangements," in *Algorithms in Bioinformatics*. Springer, 2006, pp. 163–173.

[10] M. E. Cosner, R. K. Jansen, B. M. Moret, L. A. Raubeson, L.-S. Wang, T. Warnow, S. K. Wyman *et al.*, "A new fast heuristic for computing the breakpoint phylogeny and experimental phylogenetic analyses of real and synthetic data." in *ISMB*, 2000, pp. 104–115.

[11] I. Peer and R. Shamir, "The median problems for breakpoints are np-complete," in *Elec. Colloq. on Comput. Complexity*, vol. 71, no. 5. Citeseer, 1998.

[12] A. Caprara, "Formulations and hardness of multiple sorting by reversals," in *Proceedings of the third annual international conference on Computational molecular biology*. ACM, 1999, pp. 84–93.

[13] E. Tannier, C. Zheng, and D. Sankoff, "Multichromosomal genome median and halving problems," in *Algorithms in Bioinformatics*. Springer, 2008, pp. 1–13.

[14] ——, "Multichromosomal median and halving problems under different genomic distances," *BMC bioinformatics*, vol. 10, no. 1, p. 120, 2009.

[15] M. Blanchette, G. Bourque, and D. Sankoff, "Breakpoint phylogenies," *Genome Informatics*, vol. 8, pp. 25–34, 1997.

[16] D. Sankoff, R. J. Cedergren, and G. Lapalme, "Frequency of insertion-deletion, transversion, and transition in the evolution of 5s ribosomal rna," *Journal of Molecular Evolution*, vol. 7, no. 2, pp. 133–149, 1976.

[17] D. A. Bader, B. M. Moret, and M. Yan, "A linear-time algorithm for computing inversion distance between signed permutations with an experimental study," *Journal of Computational Biology*, vol. 8, no. 5, pp. 483–491, 2001.

[18] B. M. Moret, L.-S. Wang, T. Warnow, and S. K. Wyman, "New approaches for reconstructing phylogenies from gene order data," *Bioinformatics*, vol. 17, no. suppl 1, pp. S165–S173, 2001.

[19] B. M. Moret, A. C. Siepel, J. Tang, and T. Liu, "Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data," in *International Workshop on Algorithms in Bioinformatics*. Springer, 2002, pp. 521–536.

[20] A. Caprara, "On the practical solution of the reversal median problem," in *International Workshop on Algorithms in Bioinformatics*. Springer, 2001, pp. 238–251.

[21] B. Larget, J. B. Kadane, and D. L. Simon, "A bayesian approach to the estimation of ancestral genome arrangements," *Molecular phylogenetics and evolution*, vol. 36, no. 2, pp. 214–223, 2005.

[22] S. Yancopoulos, O. Attie, and R. Friedberg, "Efficient sorting of genomic permutations by translocation, inversion and block interchange," *Bioinformatics*, vol. 21, no. 16, pp. 3340–3346, 2005.

[23] F. Ronquist and J. P. Huelsenbeck, "Mrbayes 3: Bayesian phylogenetic inference under mixed models," *Bioinformatics*, vol. 19, no. 12, pp. 1572–1574, 2003.

[24] H. Matsuda, "Protein phylogenetic inference using maximum likelihood with a genetic algorithm," in *Pacific symposium on biocomputing*, vol. 96. Citeseer, 1996, pp. 512–523.

[25] A. A. Andreatta and C. C. Ribeiro, "Heuristics for the phylogeny problem," *Journal of Heuristics*, vol. 8, no. 4, pp. 429–447, 2002.

[26] G. B. Fogel, "Evolutionary computation for the inference of natural evolutionary histories," *IEEE Connections*, vol. 3, no. 1, pp. 11–14, 2005.

[27] J. E. Gallardo, C. Cotta, and A. J. Fernández, "Reconstructing phylogenies with memetic algorithms and branch-and-bound." 2007.

[28] J.-M. Richer, A. Goëffon, and J.-K. Hao, "A memetic algorithm for phylogenetic reconstruction with maximum parsimony," in *European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics*. Springer, 2009, pp. 164–175.

[29] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[30] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi, "Variable neighborhood search: basics and variants," *EURO Journal on Computational Optimization*, pp. 1–32, 2016.

[31] M. A. M. de Oca, C. Cotta, and F. Neri, "Local search," in *Handbook of Memetic Algorithms*. Springer, 2012, pp. 29–41.

[32] M. S. Waterman and T. F. Smith, "On the similarity of dendrograms," *Journal of Theoretical Biology*, vol. 73, no. 4, pp. 789–800, 1978.

[33] C. C. Ribeiro and D. S. Vianna, "A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking." in *WOB*, 2003, pp. 97–102.

[34] ——, "A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy," *International Transactions in Operational Research*, vol. 16, no. 5, pp. 641–657, 2009.

[35] J.-M. Richer, E. Rodriguez-Tello, and K. E. Vazquez-Ortiz, "Maximum parsimony phylogenetic inference using simulated annealing," in *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*. Springer, 2013, pp. 189–203.

[36] B. L. Allen and M. Steel, "Subtree transfer operations and their induced metrics on evolutionary trees," *Annals of combinatorics*, vol. 5, no. 1, pp. 1–15, 2001.

[37] C. Cotta and P. Moscato, "Inferring phylogenetic trees using evolutionary algorithms," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2002, pp. 720–729.

[38] M. E. Cosner, R. K. Jansen, B. M. Moret, L. A. Raubeson, L.-S. Wang, T. Warnow, and S. Wyman, "An empirical comparison of phylogenetic methods on chloroplast gene order data in campanulaceae," in *Comparative Genomics*. Springer, 2000, pp. 99–121.

[39] M. Valach, Z. Farkas, D. Fricova, J. Kovac, B. Brejova, T. Vinar, I. Pfeiffer, J. Kucsera, L. Tomaska, B. F. Lang *et al.*, "Evolution of linear chromosomes and multipartite genomes in yeast mitochondria," *Nucleic acids research*, p. gkq1345, 2011.