HEURISTICS FOR THE PHYLOGENY PROBLEM*

ALEXANDRE A. ANDREATTA† AND CELSO C. RIBEIRO‡

Abstract. A phylogeny is a tree that relates taxonomic units, based on their similarity over a set of characters. The problem of finding a phylogeny with the minimum number of evolutionary steps (the so-called parsimony criterion) is one of the main problems in comparative biology. In this work, we study different heuristic approaches to the phylogeny problem under the parsimony criterion. New algorithms based on metaheuristics are also proposed. All heuristics are implemented and compared under the same framework, leading to consistent and thorough comparative results. Computational results are reported for benchmark instances from the literature.

Key words. Phylogeny problem, phylogenetic trees, evolutionary trees, parsimony, local search, heuristics, frameworks

1. Phylogeny Problem. A phylogeny (or an evolutionary tree) is a tree that relates groups of species, populations of distinct species, populations of the same species, or homologous genes in populations of distinct species, indistinctly denoted by taxons [3, 29, 30]. Such relations are based on the similarity over a set of characters. Tree leaves represent the taxons under analysis, while interior nodes represent hypothetical (unknown) ancestors.

Characters are independent attributes used to compare taxons. A character may represent e.g. a morphologial attribute or a position within a sequence of nucleotides. Each character take values on a finite set of possible states. For example, if some character describes a position in a sequence of nucleotides, then the possible states are "Adenine", "Cytosine", "Guanine", and "Timine". Each taxon is defined by their character states. Binary characters are those who have only two possible states, which represent the presence or the absence of some attribute. Instances of the phylogeny problem (with binary characters) are then characterized by 0-1 matrices, in which each element (i, j) corresponds to the state of character j within taxon i. Each row represents the characteristic vector of some taxon.

Different criteria may be used in the evaluation of phylogenetic trees. Each state change along a branch of the phylogenetic tree is counted as an evolutionary step. The parsimony criterion states that the best phylogeny is the one that can be explained by the minimum number of evolutionary steps. It is frequently said that this criterion is the best one if the probability of character mutation is very small [19, 21, 28].

Given a set of taxons defined by a set of characters, the phylogeny problem (under parsimony) consists in finding a phylogeny with the minimum number of evolutionary steps. The phylogeny problem is one of the main problems in comparative biology. It is NP-hard in general and in the common restricted cases [4, 6, 13, 14].

---

*July 7, 2000

†Department of Applied Computer Science, University of Rio de Janeiro - UNIRIO, Rua Voluntários da Pátria 107, Rio de Janeiro, RJ 22270, Brazil. E-mail: `andreatt@uniriotec.br`

‡Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, RJ 22453-900, Brazil. E-mail: `celso@inf.puc-rio.br`

	a	b	c	d	e	f	g
O	0	0	0	0	0	0	0
A	1	1	0	0	1	1	1
B	1	1	1	1	1	1	1
C	1	1	1	1	0	0	0

TABLE 1.1
Instance of the phylogeny problem.

Table 1.1 illustrates an instance of the phylogeny problem defined by a set of taxons $\{O, A, B, C\}$ and binary characters $\{a, b, c, d, e, f, g\}$. Figures 1.1 and 1.2 illustrate two alternative optimal solutions for this instance, both with nine evolutionary steps. We indicate on each branch of these trees the characters which change from one taxon to another, together with the associated number of evolutionary steps.

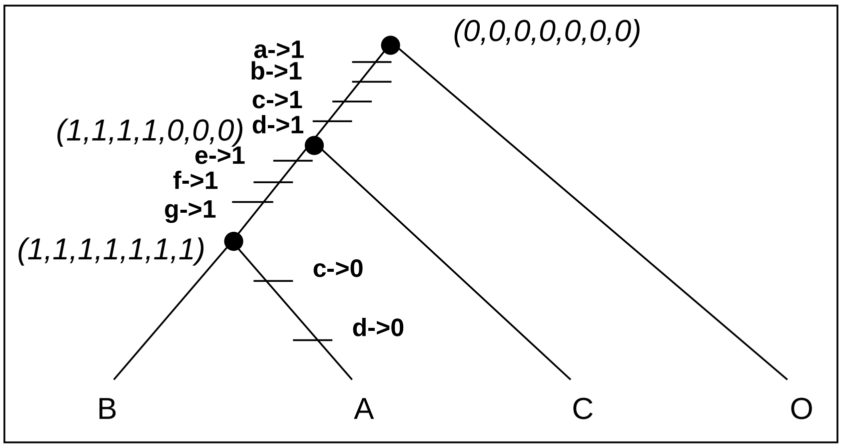

FIG. 1.1. *Optimal phylogeny with state reversions.*

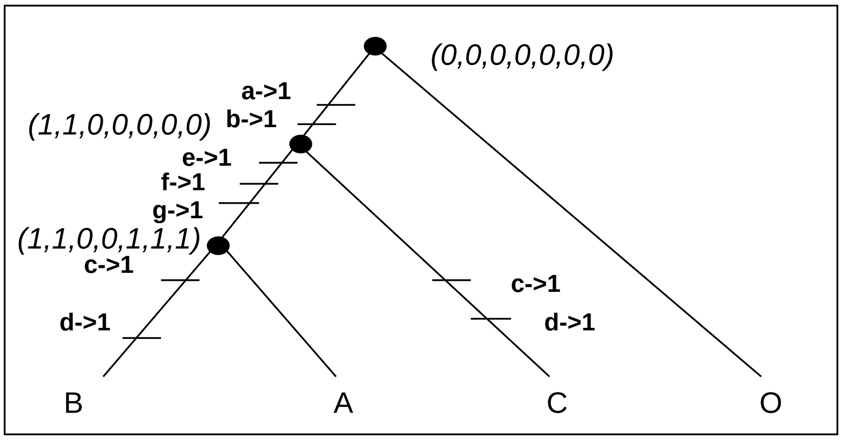

FIG. 1.2. *Optimal phylogeny with parallel derivations.*

Although the phylogeny problem can be modeled as a minimum Steiner tree problem in a metric space [27], the formulation below is more appropriate for algorithmic developments. Let E be the set of all possible taxons (Cartesian product of the character domains) and $W = \{x^{(1)}, \ldots, x^{(n)}\} \subseteq E$ the set of operational taxons (those under analysis). A phylogenetic tree s for the operational taxons W belongs to the set S of all unrooted tree with n leaves (each leave is in one-to-one correspondence with an operational taxon $x^{(i)} \in W, i = 1, \ldots, n$) and all internal nodes with degree three. Let $f : S \rightarrow \mathbb{R}$ be a function which associates each phylogeny $s \in S$ to its parsimony value. The phylogeny problem under parsimony may then be formulated as that of finding a phylogeny $s^\star \in S$ such that $f(s^\star) = \min_{s \in S} f(s)$. Farris [8], Fitch [11], and Fitch and Farris [12] proposed polynomial algorithms running in $O(mn)$ time for the computation of the parsimony values for a given phylogeny, where n is the number of operational characters and m the number of binary characters.

Several heuristic or exact approaches (for small problems) for the computation of phylogenetic trees are dispersed through the scientific literature (see e.g. [7, 22, 23, 24]), many of them in Biology journals in which they are not properly described in algorithmic terms. The variety of existing methods require careful comparative studies regarding their effectiveness and considering the same test problems and computational environment. In this work, we develop a thorough comparative study of existing and new heuristics for the phylogeny problem. In the next section, we present an overview of the **Searcher** framework, designed to provide an architectural basis for the implementation and comparison of local search strategies. In the remaining of the paper, we show how this framework can be used to systematize the evaluation and the comparison of construction, local search, and metaheuristic algorithms for the phylogeny problem. In Section 3, several algorithms for the construction of phylogenies are introduced. Neighborhood structures are discussed in Section 4, while Section 5 is devoted to basic local search approaches. Metaheuristic strategies are proposed in Section 6. All these approaches have been implemented using the **Searcher** framework. Computational results obtained for benchmark instances of the phylogeny problem are presented and discussed in detail. Concluding remarks are drawn in the final section.

2. The Searcher Framework. The **Searcher** framework was designed to provide an architectural basis for the implementation and comparison of local search strategies [1, 2]. Frameworks are essentially prototypical architectures for related systems, or systems in the same application domain [5, 9, 15, 20, 26]. Their strength lies in the fact that they offer abstract solutions to problems within the domain. These solutions can be instantiated, for example, with the refinement of the abstract classes in the framework and with the particularization of generic classes. Frameworks fundamentally differ from class libraries, in that they offer more than resources to use in a given situation. Framework classes are bound to one another via relationships also defined as part of the framework.

Searcher models in separate (and related) classes the different concerns involved in local search, increasing the ability to construct and compare heuristics. Several aspects are encapsulated, such as algorithms for the construction of initial solutions, methods for neighborhood generation, and movement selection criteria. This classification simplifies implementations and invites new extensions, in the form of subclasses. Code reuse allows a better platform for comparison studies, since a large part of the code is common to all members of the family. The **Searcher** framework can be used and has particularly appropriate features for situations involving:

(a) local search strategies that use different methods for the construction of the initial solution, different neighborhood relations, or different movement selection criteria;
(b) construction algorithms that utilize subordinate local search heuristics for the improvement of partially constructed solutions; and
(c) local search heuristics with dynamic neighborhood models.

Figure 2.1 shows the classes and relationships involved in the **Searcher** framework. The OMT notation [25] for static view is used. The *Client* wants a *Solution* for a problem instance. It delegates this task to its *SearchStrategy*, which is composed by at least one *BuildStrategy* and one *LocalSearch*. The *BuildStrategy* produces an initial *Solution* and the *LocalSearch* improves the initial *Solution* through successive movements. The *BuildStrategy* and the *LocalSearch* perform their tasks based on neighborhood relations provided by the *Client*. The implementation of these neighborhoods is delegated by the *Solution* to its *IncrementModel* (related to the *BuildStrategy*) and to its *MovementModel* (related to the *LocalSearch*). The *IncrementModel* and the *MovementModel* are the objects that obtain the *Increments* or the *Movements* necessary to modify the *Solution* (under construction or not). The *MovementModel* may change at run-time, reflecting the use of a dynamic neighborhood in the *LocalSearch*. The variation of the *MovementModel* is controlled inside the *LocalSearch*.

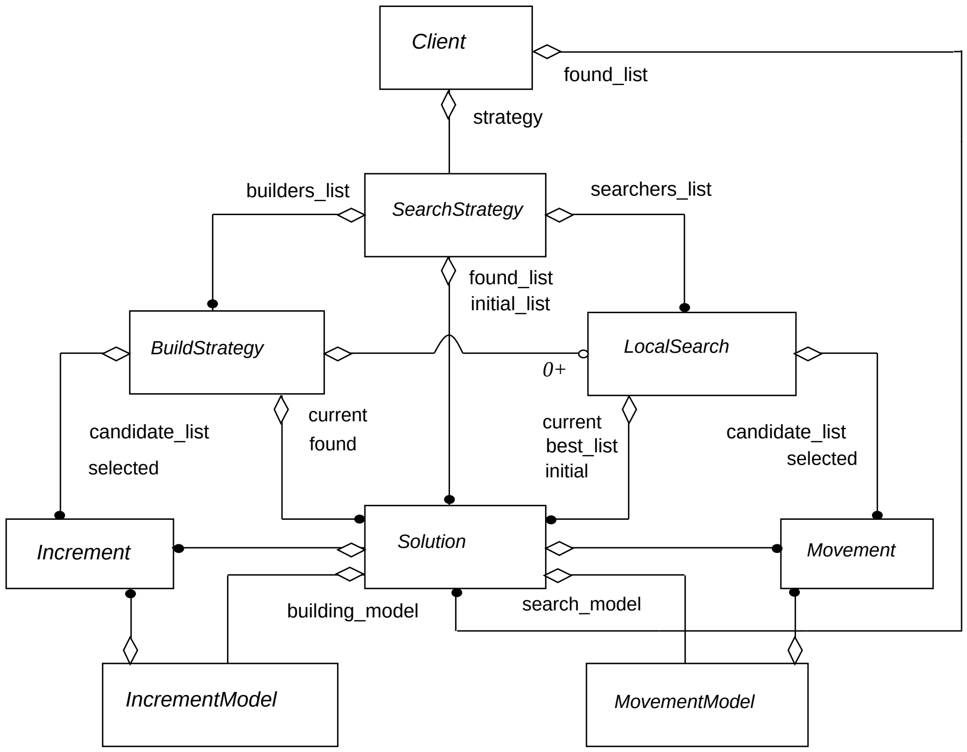

FIG. 2.1. *Class diagram of the* **Searcher** *framework.*

We now comment some implementation issues. *SearchStrategy* is a purely virtual class, where the methods to instantiate concrete *BuildStrategys* and *LocalSearchs* are defined. We implemented a concrete subclass with one *BuildStrategy* and one *Lo-*

calSearch. To implement the construction algorithms presented in Section 3, we created two concrete *BuildStrategys*. Both of them allow using subordinate *LocalSearchs* to improve partial solutions along the construction.

We implemented three *MovementModels*, supporting the three different neighborhood relationships (NNI, STEP, and SPR) defined in Section 4. Coding such *MovementModels* in accordance with the relationships defined by the **Searcher** framework, made possible the implementation and test of constructive algorithms using subordinate iterative improvement procedures to improve partial phylogenies. With respect to the iterative improvement procedures described in Section 5, we created only one concrete *LocalSearch*, which uses two move selection criteria: first improving move and best improving move. Neighborhoods NNI, STEP, and SPR have been implemented as three subclasses of *MovementModel*.

To implement GRASP, it was just a matter of creating a concrete *SearchStrategy* to start up a constructive algorithm several times and apply an iterative improvement procedure to each solution obtained. The multistart variant of the original VNS metaheuristic described in Section 6 was implemented as a concrete *SearchStrategy*.

Besides allowing code reuse, the use of the framework gave flexibility in programming and enabled the fast development of different algorithms, variants, and combinations. The **Searcher** framework addresses the most important design issues, which are how to separate responsibilities and how the agents of these tasks are related.

All algorithms developed in this work under the **Searcher** framework have been implemented in C++ and compiled with the SunSoft SPARCompiler C++ 4.1 with default optimization options. The choice of the C++ language for its implementation was motivated by the need to generate efficient code to manage problem instances of great size. Moreover, although C++ supports object-oriented programming, it is possible to relax the paradigm whenever one needs more speed (relaxing the data encapsulation restriction, for instance). We have also used the generic class library Tools.h++ from Rogue Wave Software. The computational experiments which will be reported in Sections 3, 5, and 6 have been performed on a Sun SparcStation 4.

3. Construction Algorithms. The basic construction algorithm for the phylogeny problem is described in Figure 3.1. A phylogeny $s \in S$ relating the taxons in W is built in $n = |W|$ iterations. In the k-th iteration, a partial phylogeny $s^{(k)}$ (defined on a subset $U \subset W$ of operational taxons) is modified by the introduction of a taxon $t \in W \setminus U$. The algorithm stops when $U = W$. Variants of this algorithm differ by the criteria they use to select a new taxon to be inserted and by the way in which $s^{(k)}$ is modified to obtain $s^{(k+1)}$. The set of modifications applied to $s^{(k)}$ to reach $s^{(k+1)}$ is called an increment. We define by $\delta(s^{(k)}, s^{(k+1)}) = f(s^{(k+1)}) - f(s^{(k)})$ the increase in the cost function $f(.)$ due to the increment leading from $s^{(k)}$ to $s^{(k+1)}$.

Increments are defined by the insertion of a taxon into a branch of the current partial solution, as illustrated in Figure 3.2. In this case, there are three possible alternatives for the insertion of taxon D into a partial solution formed by three taxons A, B, and C. A single incremental step is defined by three attributes: (1) taxon to be inserted, (2) branch into which this taxon will be inserted, and (3) a nonnegative incremental value $\delta(s^{(k)}, s^{(k+1)})$.

Luckow and Pimentel [22] have conducted the first evaluation study of algorithms

```
procedure Build-phylogeny
    s ← ∅; U ← ∅; k ← 0
    loop
        exit when (k = n)
        Select a taxon t ∈ W\U.
        U ← U ∪ {t}
        Modify s by inserting t.
        k ← k + 1
    end-loop
end-procedure
```

FIG. 3.1. *Basic construction algorithm.*

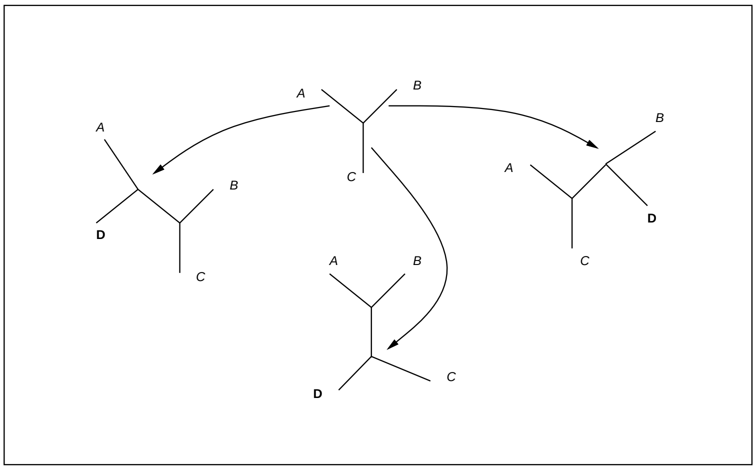

FIG. 3.2. *Insertion alternatives of a new taxon into a phylogeny with three taxons.*

for the phylogeny problem, in which they tested strategies offered by some software packages. Another study was later conducted by Platnick [23], involving other programs. These studies do not provide a good basis for assessing algorithm efficiency and their results are not conclusive, as far as the algorithms themselves are not precisely described in the references, and completely different techniques and languages have been used for their implementation.

In this work, we have implemented and compared under the **Searcher** framework several variants of the basic construction algorithm, as described below. The complexity of the evaluation of each single step incremental value $\delta(s^{(k)}, s^{(k+1)})$ is $O(mk)$, where m is the number of binary characters and k the number of taxons in the current partial phylogeny.

• **1stRstep** (first random step): the taxon to be inserted and the branch where the insertion will take place are randomly selected according with a discrete uniform probability distribution. Since the selection procedure has $O(1)$ complexity, the whole construction procedure has $\sum_{k=1}^{n} O(mk)O(1) = O(mn^2)$ time complexity.

• **1stRotuGbr** (first random operational taxon unit insertion over greedy branch): a taxon randomly selected according with a discrete uniform probability distribution is inserted in the first branch leading to the minimum increase in the parsimony value. Since there are $2k-5$ possible branches for the insertion of a given taxon and each increment evaluation is performed in $O(mk)$, the overal complexity of this algorithm is $\sum_{k=3}^{n} O(mk)(2k-5) = O(mn^3)$.

• **Gstep_wR** (greedy step with randomness): a pair taxon-branch is randomly selected from among all those leading to the most parsimonius increment value. Since there are still $n-(k-1)$ unselected taxons in iteration k, and $2k-5$ possible branches for each insertion, whose increment evaluation is performed in time $O(mk)$, the overall complexity is $\sum_{k=3}^{n} O(mk)(2k-5)[n-(k-1)] = O(mn^4)$.

• **GRstep** (greedy randomized step): this greedy randomized algorithm differs very slightly from the previous one. Each increment is selected from a restricted candidate list which contains not only optimal parsimony increments, but also all those defined by pairs taxon-branch whose value is at most 10% greater than the minimum value. Its complexity is the same of the previous algorithm, but the algorithm is slower due to the larger size of the restricted candidate list.

Eight benchmark real-life test instances [17, 22, 23, 24] have been considered for the evaluation of the algorithms discussed in this work (seven of them are available from http://www.vims.edu/~mes/hennig/hennig.html). For each test instance we report in Table 3.1 its identification, the number of taxons (n), the number of characters (m), and the parsimony value of the best known solution. ANGI and TENU have only binary characters, while the others present some multi-state characters dealt with as linear ones and converted to equivalent binary characters. Characters of the instance GOLO are positions in a sequence of nucleotides, while the others are morphological characters.

Instance	Taxons (n)	Characters (m)	Best value
GRIS	47	93	172
ANGI	49	59	216
TENU	56	179	682
ETHE	58	86	372
ROPA	75	82	326
GOLO	77	97	497
SCHU	113	146	760
CARP	117	110	548

TABLE 3.1
Test instances.

We summarize below some computational results derived from 100 runs of each instance. We use the relative deviation $D(s) = ((f(s) - f^\star)/f^\star) \times 100$ of a given phylogeny s from the best known parsimony value $f^\star$ as a normalized indication of solution quality. Detailed results for all test instances and algorithms can be found in Andreatta [1]. Table 3.2 illustrates the computational results obtained for instance SCHU. For each algorithm, t_a denotes the average computation time (in seconds) over the 100 runs, while D_m, D_a, and D_M denote respectively the minimum, average, and maximum relative deviations with respect to the best known parsimony value.

For each algorithm, Table 3.3 reports average results t_a^{avg}, D_m^{avg}, D_a^{avg}, and D_M^{avg} over the eight test instances.

In order to give a more clear picture of the behavior of each algorithm, we report

	1stRstep	1stRotuGbr	Gstep_wR	GRstep
t_a	1.0	7.7	104.3	139.6
D_m	217.8%	3.3%	2.5%	10.5%
D_a	239.3%	5.9%	3.9%	20.7%
D_M	264.7%	9.3%	5.9%	26.4%

TABLE 3.2
Results of construction algorithms for instance SCHU.

	1stRstep	1stRotuGbr	Gstep_wR	GRstep
t_a^{avg}	0.45	8.43	46.41	59.89
D_m^{avg}	128.9%	2.1%	2.7%	10.1%
D_a^{avg}	144.0%	5.6%	4.8%	17.7%
D_M^{avg}	157.2%	10.0%	8.2%	25.3%

TABLE 3.3
Average results of construction algorithms over the eight instances.

in Figure 3.3 detailed results obtained with the four algorithms for instance SCHU. In the upper part of this figure, we report the distribution of the parsimony values obtained by each algorithm along the 100 runs. Algorithm 1stRstep leads to very bad solutions with a large variance, as expected from a purely random algorithm. Algorithms 1stRotuGbr and Gstep_wR lead to small variances and the best solutions overall. The results obtained by algorithm GRstep show larger variance and worse solutions than the latter. In the lower part of Figure 3.3, we report minimum, average, and maximum computation times over the 100 runs (in seconds), as well as minimum, average, and maximum relative deviations with respect to the best known solution observed for each algorithm over the 100 runs. We notice that although Gstep_wR finds better solutions than 1stRotuGbr, it is approximately 10 times slower than the latter.

To conclude, algorithms 1stRotuGbr and Gstep_wR seem to be the most effective in terms of both solution quality and computation time. Algorithm 1stRotuGbr leads to good solutions in small computation times, while algorithm Gstep_wR finds the best solutions in computation times one order of magnitude higher. The maximum relative deviations with respect to the best known solution over the eight test instances for 1stRotuGbr and Gstep_wR were 14.8% and 16.9%, respectively. Accordingly, these two algorithms will be used in Section 5 for the construction of starting solutions for local search procedures.

4. Neighborhoods. Local search or neighborhood improvement methods are based on the investigation of the solution space, by successively exploring the neighborhood of a current solution and criteriously moving to one of its neighbors. We describe three neighborhood relations [29] for the phylogeny problem, based on the formulation presented in Section 1. We denote by s any current solution whose neighborhood is under investigation, and by s' one of its neighbors. We also recall that n denotes the number of operational taxons, while m is the number of characters under observation.

• **Nearest Neighborhood Interchanges** (NNI): This neighborhood is defined by the permutation of subtrees around internal branches (i.e., those connecting internal

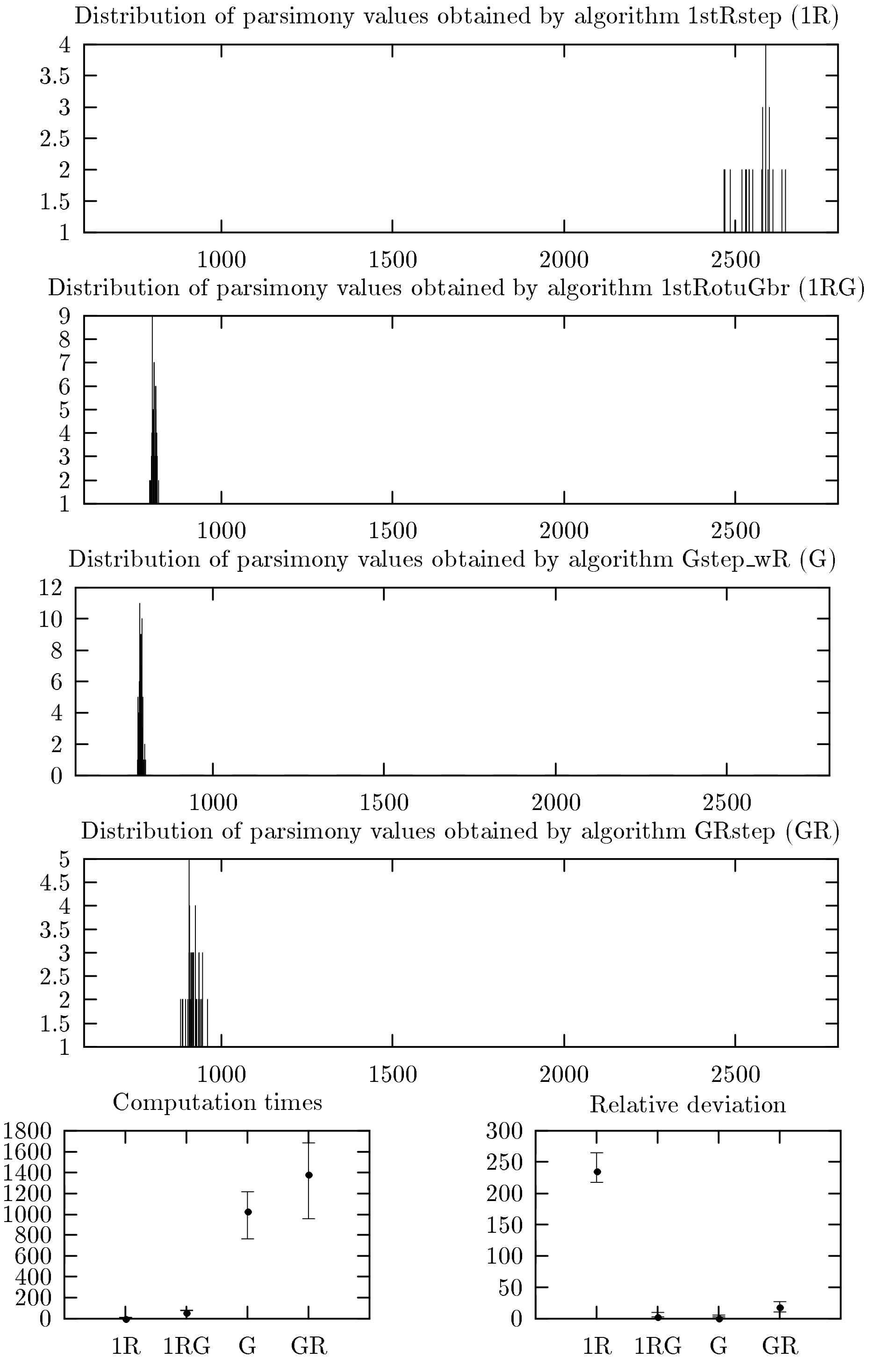

FIG. 3.3. *Detailed results for instance SCHU with construction algorithms (100 runs).*

nodes). Figure 4.1 illustrates an example of a move within this neighborhood (subtrees A, B, C, and D may also be leaves). There are two possible neighbors for each internal branch. Since each phylogeny has $n-3$ internal branches, it has $2n-6$ neighbors.

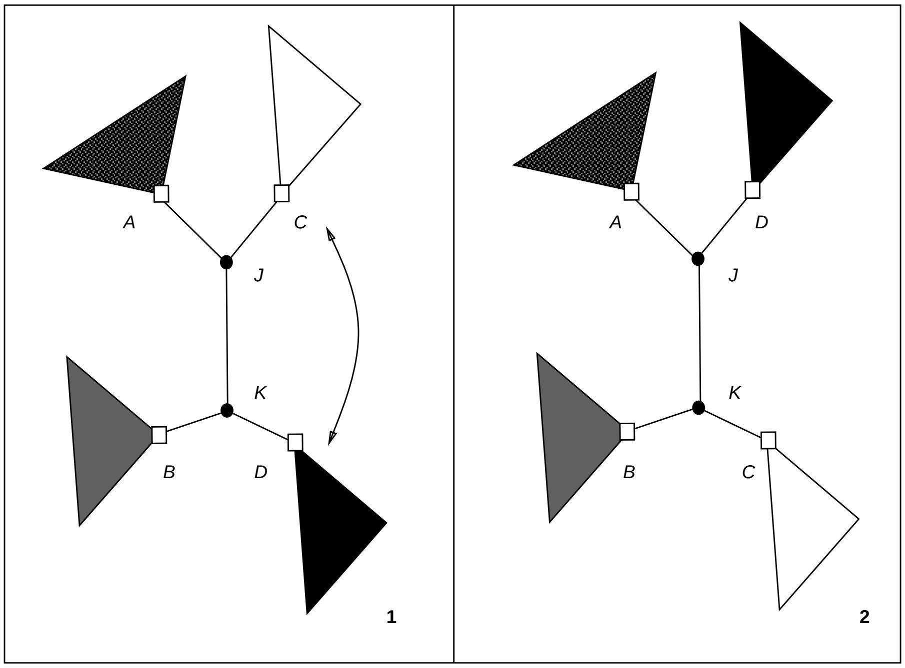

FIG. 4.1. *Move within neighborhood NNI.*

• **Single Step** (STEP): According with this neighborhood relation, a neighbor is obtained by taking out a taxon (i.e., a leave) from the current solution and putting it back into another branch of the tree. Figure 4.2 illustrates an example of a move within this neighborhood. Since each taxon may be reinserted into $2(n-1)-3-1$ different branches, each solution has $2n(n-3)$ neighbors.

• **Subtree Pruning and Regrafting** (SPR): Solutions in this neighborhood are obtained by eliminating one internal node and its three adjacent branches. Next, two pending nodes are joined by a new branch. The still pending subtree is reconnected by its pending node to a branch of the other subtree. Figure 4.3 illustrates an example of a move within this neighborhood, which contains the previous one and has no more than $4(n-3)(n-2)$ solutions [1].

As mentionned in Section 1, efficient polynomial algorithms exist for the computation of the parsimony value of a given phylogeny [8, 11, 12]. However, for all neighborhood definitions, these algorithms can be further improved to fastly recompute the cost of a neighbor solution directly from that of the current solution, instead of computing it from scratch [1, 16].

5. Local Search. In this section we explore local search procedures for the phylogeny problem. We first investigate different variants of the iterative improvement procedure for local search. Next, we explore a dynamic neighborhood model to construct an iterative improvement procedure based on variable neighborhoods.

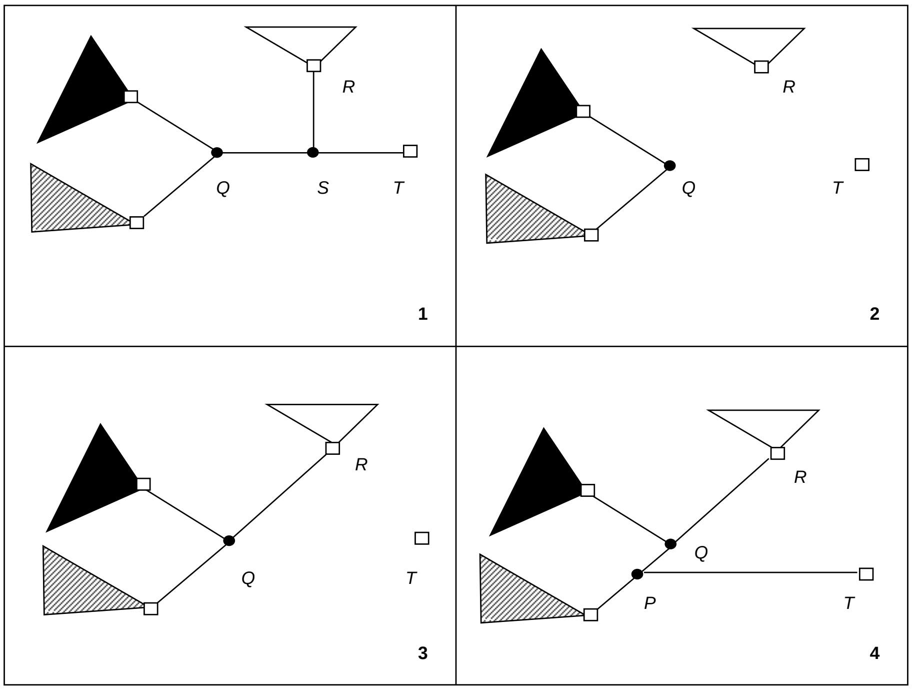

FIG. 4.2. *Move within neighborhood STEP.*

5.1. Iterative Improvement. The basic iterative improvement algorithm for local search is given in Figure 5.1. Different variants can be built according with the neighborhood structure (NNI, STEP, and SPR; see Section 4), the move selection criterion (first improving move, best improving move), and the algorithm used to build the initial solution. Once again, these algorithms have been implemented under the **Searcher** framework described in Section 2.

We summarize below some computational results derived from 100 runs of each instance. Detailed results for all test instances and variants of the basic algorithm can be found in Andreatta [1]. Table 5.1 illustrates the computational results obtained for instance SCHU. For each combination of neighborhood structure, move selection criterion, and construction algorithm, T_a (resp. t_a) denotes the average computation time (in seconds) over the 100 runs, including (resp. not including) the construction of the initial solution, while D_m, D_a, and D_M denotes respectively the minimum, average, and maximum relative deviations with respect to the best known parsimony value. We also give in this table the value w_a, which represents the average difference between the parsimony value of the initial and final solutions, divided by the value of the best known solution. The small values observed for w_a in the case of neighborhood NNI mean that most of the optimization effort has been performed by the construction algorithm, not by the local search. The reduced computation times observed for neighborhood NNI reflect the fact this is a poor neighborhood model.

Table 5.2 reports average results over the eight test instances. T_a^{avg} denotes the average computation time (in seconds) over the 100 runs, including the construction of the initial solution, while D_a^{avg} denotes the average relative deviation with respect

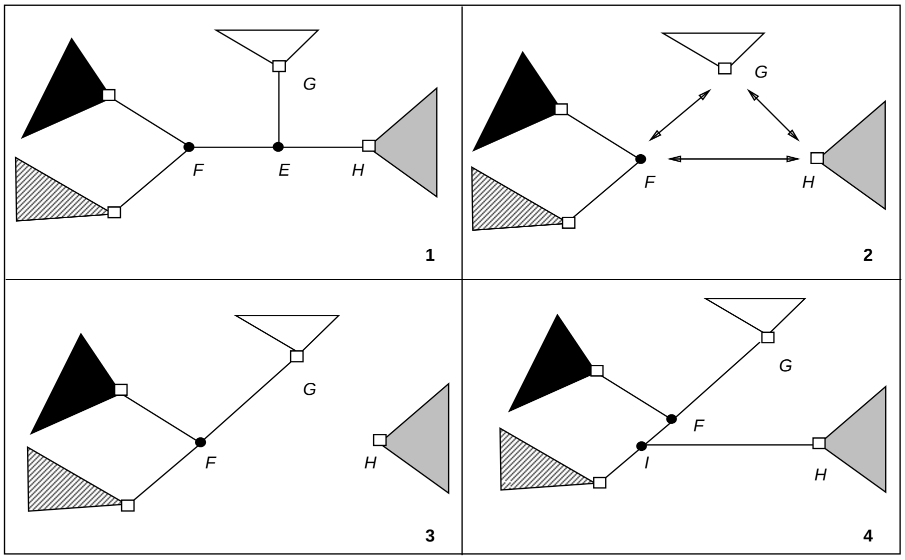

FIG. 4.3. *Move within neighborhood SPR.*

procedure Iterative-improvement
  Let $s_0 \in S$ be the starting solution.
  $s \leftarrow s_0$
  loop
    exit when $(\nexists s' \in N(s) : \delta(s, s') < 0)$
    Pick any $s' \in N(s) : \delta(s, s') < 0$.
    $s \leftarrow s'$
  end-loop
end-procedure

FIG. 5.1. *Iterative improvement algorithm for local search.*

to the best known parsimony value.

We notice from Table 5.2 that strategies based on selecting the first improving move are better than those using the best improving move, since computation times are significantly smaller for the first, while the average solution quality is approximately the same. The iterative improvement strategy [Gstep_wR,1stImpr,SPR] defined by the combination of the construction algorithm Gstep_wR with neighborhood SPR for local search leads to the best solutions in terms of their parsimony values.

5.2. Variable Neighborhoods. Let $N^{(1)}$, $N^{(2)}$, ..., $N^{(r)}$ be different neighborhood structures for some combinatorial optimization problem. In the case of the phylogeny problem, we take $r = 3$ and $N^{(1)}$ = NNI, $N^{(2)}$ = STEP, and $N^{(3)}$ = SPR. Variable neighborhood methods are based on exploring neighborhood changes, determined by specific events occured during the search.

We investigate an iterative improvement procedure based on variable neighborhoods, which essentially is a variant of the Variable Neighborhood Descent (VND) approach proposed by Mladenović and Hansen [18]. Basically, the algorithm performs an iterative improvement procedure using the first neighborhood $N^{(1)}$. Whenever a

Neighborhood NNI	Algorithm 1stRotuGbr		Algorithm Gstep_wR	
	Best	First	Best	First
T_a	8.2	8.0	104.7	104.6
t_a	0.5	0.3	0.4	0.3
D_m	2.8%	2.9%	1.4%	1.4%
D_a	4.8%	4.9%	3.1%	3.1%
D_M	7.5%	7.8%	4.7%	4.7%
w_a	1.0 %	1.0%	0.8%	0.8 %

Neighborhood STEP	Algorithm 1stRotuGbr		Algorithm Gstep_wR	
	Best	First	Best	First
T_a	127.7	75.8	242.5	197.0
t_a	120.1	68.1	138.2	92.7
D_m	2.2%	2.2%	0.7%	0.7%
D_a	4.3%	4.3%	2.2%	2.3%
D_M	7.1%	7.2%	3.7%	3.7%
w_a	1.6%	1.5%	1.7%	1.6%

Neighborhood SPR	Algorithm 1stRotuGbr		Algorithm Gstep_wR	
	Best	First	Best	First
T_a	1134.7	697.9	1197.3	732.1
t_a	1127.0	690.2	1093.0	627.8
D_m	0.4%	0.4%	0.3%	0.4%
D_a	2.3%	2.3%	1.1%	1.4%
D_M	4.7%	4.3%	2.4%	3.2%
w_a	3.6%	3.5%	2.8%	2.5%

TABLE 5.1
Results of local search algorithms for instance SCHU.

		Algorithm 1stRotuGbr		Algorithm Gstep_wR	
		Best	First	Best	First
NNI	T_a^{avg}	4.4	4.3	46.7	46.6
	D_a^{avg}	4.5%	4.5%	4.0%	4.0%
STEP	T_a^{avg}	59.4	38.7	109.2	88.0
	D_a^{avg}	3.8%	3.8%	2.8%	2.9%
SPR	T_a^{avg}	446.3	287.6	518.0	289.1
	D_a^{avg}	2.1%	2.2%	1.7%	1.9%

TABLE 5.2
Average results of local search algorithms for the eight instances.

local optimum is found, the current neighborhood is changed. In case an improving move is found, the algorithm continues the search exploring the first neighborhood. The algorithm stops when a local optimum is found with the last neighborhood structure. Figure 5.2 gives the algorithmic desciption of procedure VND which implements this approach.

```
procedure VND
    Let s_0 be an initial solution.
    s ← s_0
    k ← 1
    loop
        exit when (k > r)
        Find the best solution s' ∈ N^(k)(s).
        if (f(s') < f(s))
            s ← s'
            k ← 1
        else k ← k + 1
        end-if
    end-loop
end-procedure
```

FIG. 5.2. *Iterative improvement with variable neighborhoods.*

	GRIS		ANGI		TENU		ETHE	
	VND	SPR	VND	SPR	VND	SPR	VND	SPR
T_a	57.3	82.2	43.4	52.7	180.8	280.4	68.5	68.5
t_a	44.7	69.7	30.6	39.9	127.9	228.0	51.4	51.4
D_m	0.0%	0.0%	0.4%	0.4%	0.1%	0.0%	1.0%	0.8%
D_a	1.7%	2.8%	2.3%	2.3%	0.6%	0.6%	1.8%	1.8%
D_M	5.2%	7.0%	4.5%	4.5%	1.6%	1.5%	3.2%	3.2%
w_a	8.7%	7.6%	2.3%	2.3%	1.5%	1.5%	1.4%	1.4%

	ROPA		GOLO		SCHU		CARP	
	VND	SPR	VND	SPR	VND	SPR	VND	SPR
T_a	128.8	204.6	189.1	350.5	436.4	732.1	441.3	541.5
t_a	93.0	168.8	136.4	297.7	332.1	627.8	357.6	457.7
D_m	0.6%	0.6%	1.4%	1.6%	0.3%	0.4%	1.0%	0.7%
D_a	2.0%	2.1%	3.4%	3.8%	1.2%	1.4%	2.5%	2.7%
D_M	3.6%	3.7%	5.8%	6.2%	2.4%	3.2%	5.0%	5.0%
w_a	3.0%	2.9%	3.7%	3.3%	2.7%	2.5%	4.0%	3.8%

TABLE 5.3
Results obtained with VND for the eight instances.

Algorithm Gstep_wR is used for the construction of the initial solution. The first improving move is selected at each local search iteration. Comparative results for 100 runs of VND and [Gstep_wR,1stImpr,SPR] (denoted by SPR) for each instance are presented in Table 5.3. We notice that the computation times observed with VND are always significantly smaller than those observed with the iterative improvement strategy [Gstep_wR,1stImpr,SPR]. Moreover, VND finds better solutions in the average than the latter, as it can be concluded by the comparison of the average relative deviations with respect to the best known solutions.

6. Metaheuristics. In this section, we investigate metaheuristic implementations based on two different paradigms: GRASP and VNS. Once again, these algorithms have been implemented under the **Searcher** framework described in Section 2.

6.1. Greedy Randomized Adaptive Search Procedures. A Greedy Randomized Adaptive Search Procedure (GRASP) [10] is an iterative process, in which each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored by local search. The best solution over all GRASP iterations is returned as the result. In the construction phase, a feasible solution is built, one element at a time. At each construction iteration, the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function that estimates the benefit of selecting each element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but usually not the top one.

We use algorithm Gstep_wr described in Section 3 as a greedy randomized construction algorithm. The local search phase is implemented with the selection of the first improving move. Two neighborhood strategies are used to devise two alternative GRASP's for the phylogeny problem: the static neighborhood SPR and the dynamic neighborhood model used within the VND procedure described in Section 5.2. Numerical results reporting computation times and minimum deviations obtained with 100 GRASP iterations for the eight test instances are presented in Table 6.1.

6.2. Variable Neighborhood Search. We investigate the implementation of a multistart variant of the Variable Neighborhood Search (VNS) metaheuristic proposed by Mladenović and Hansen [18], also based on the exploration of a dynamic neighborhood model. As in Section 5.2, let $N^{(1)}$, $N^{(2)}$, $\ldots$, $N^{(r)}$ be different neighborhood structures for some combinatorial optimization problem ($N^{(1)}$ = NNI, $N^{(2)}$ = STEP, and $N^{(3)}$ = SPR for the phylogeny problem).

Basically, our VNS variant described in Figure 6.1 is a multistart strategy, whose external loop controls the number of initial solutions to be constructed and explored. At each iteration of the inner loop, we randomly select a neighbor s' of the current solution s within the current neighborhood $N^{(k)}$, which is submitted to an iterative improvement procedure. If the so found local optimum $\bar{s}$ improves the current solution s, the search is resumed from $\bar{s}$ using neighborhhod $N^{(1)}$. Otherwise, we change to the next neighborhood and repeat the previous step from the same current solution.

```
procedure VNS
    loop
        exit when (number of iterations)
        Let s0 be an initial solution.
        s ← s0
        k ← 1
        loop
            exit when (k > r)
            Randomly select s' ∈ N^(k)(s).
            s̄ ← Iterative-improvement(s')
            if (f(s̄) < f(s))
                s ← s̄
                k ← 1
            else k ← k + 1
            end-if
        end-loop
    end-loop
end-procedure
```

FIG. 6.1. *Variable neighborhood search procedure.*

Instance	Strategy	Time (min's)	Deviation (%)
GRIS	GRASP (SPR)	137.0	0.0
	GRASP (VND)	95.5	0.0
	VNS	234.7	0.0
ANGI	GRASP (SPR)	87.8	0.4
	GRASP (VND)	72.3	0.4
	VNS	195.2	0.0
TENU	GRASP (SPR)	467.3	0.0
	GRASP (VND)	300.5	0.1
	VNS	815.2	0.1
ETHE	GRASP (SPR)	114.2	0.8
	GRASP (VND)	114.2	1.0
	VNS	287.2	0.8
ROPA	GRASP (SPR)	341.0	0.6
	GRASP (VND)	214.7	0.6
	VNS	557.3	0.3
GOLO	GRASP (SPR)	584.2	1.6
	GRASP (VND)	315.2	1.6
	VNS	762.0	1.6
SCHU	GRASP (SPR)	1220.2	0.4
	GRASP (VND)	727.3	0.3
	VNS	1587.0	0.0
CARP	GRASP (SPR)	902.5	0.7
	GRASP (VND)	735.5	1.0
	VNS	1276.8	1.0

TABLE 6.1
Final results obtained with metaheuristics GRASP and VNS.

Computational results obtained with 100 VNS iterations are also presented in Table 6.1. Iterative improvement is performed using algorithm VND described in Section 5.2. Computation times are higher than those observed with the implementations of GRASP, but there are some improvements in terms of solution quality in three out of the eight test problems (instances ANGI, ROPA, and SCHU).

7. Concluding Remarks. In this work, we have studied and compared different heuristics to the phylogeny problem under the parsimony criterion. New algorithms based on VNS and GRASP metaheuristics have been proposed. Two variantes of GRASP are considered, using static and dynamic neighborhood models. The first of them is one of the most used techniques available within software packages for the phylogeny problem (although never referenced as a GRASP). All heuristics have been implemented and compared under the **Searcher** framework, which gave us flexibility to design, implement, and test many heuristics and variants, with code reuse and reduced development time.

Computational results have been reported for eight benchmark instances from the literature. We have identified two efficient construction algorithms (1stRotuGbr and Gstep_wR), with good trade-offs in terms of solution quality and computation time. The additional time taken by Gstep_wR for the construction of initial solutions is quite useful, since it leads to better solutions and comparable computation times after local

Strategy	Deviation (%)	Best values	Time (min's)
GRASP (SPR)	0.6	5	481.8
GRASP (VND)	0.6	2	321.9
VNS	0.5	6	714.4

TABLE 7.1
Final average results.

search is performed from the solutions it found. Three neighborhood structures have been compared, showing that the one with greater size performed better than the others in terms of solution quality.

We summarize average computational results obtained with the three metaheuristic implementations in Table 7.1. The three strategies obtain equivalent results in terms of the average solution quality (relative deviation from the best known solution) and the number of instances for which the best value was attained. We are currently working with more refined implementations, data structures, move evaluation algorithms, neighborhood structures, and parameter settings which should strongly improve the performance of the implementations not only of the above metaheuristics, but also of tabu search and evolutionary algorithms.

REFERENCES

[1] A.A. ANDREATTA, *A framework for the development of local search heuristics with an application to the phylogeny problem* (in Portuguese), Ph.D. Dissertation, Computer Science Department, Catholic University of Rio de Janeiro, Rio de Janeiro, 1998.

[2] A.A. ANDREATTA, S.E.R. CARVALHO, and C.C. RIBEIRO, "A framework for local search heuristics for combinatorial optimization problems", submitted for publication, 1998.

[3] F.J. AYALA, "The myth of Eve: Molecular biology and human origins", *Science* 270 (1995), 1930–1939.

[4] H. BODLAENDER, M. FELLOWS, and T. WARNOW, "Two strikes against the perfect phylogeny problem", *Proceedings of the International Conference on Algorithms, Languages and Programming*, 273–283, Springer-Verlag, 1992, Wien.

[5] F. BUSCHMANN, R. MEUNIER, H. ROHNERT, and P. SOMMERLAD, *Pattern-oriented software development*, Wiley, 1996, New York.

[6] W.H.E. DAY, D.S. JOHNSON, and D. SANKOFF, "The computational complexity of inferring rooted phylogenies by parsimony", *Mathematical Biosciences* 81 (1986), 33–42.

[7] A. DRESS and M. KRÜGER, "Parsimonious phylogenetic trees in metric spaces and simulated annealing", *Advances in Applied Mathematics* 8 (1987), 8–37.

[8] J.S. FARRIS, "Methods for computing Wagner trees", *Systematic Zoology* 19 (1970), 83– 92.

[9] M. FAYAD and D. SCHMIDT, "Object-oriented application frameworks", *Communications of the ACM* 40 (1997), 32–38.

[10] T.A. FEO and M.G.C. RESENDE, "Greedy randomized adaptative search procedures", *Journal of Global Optimization* 6 (1995), 109–133.

[11] W.M. FITCH, "Towards defining the course of evolution: Minimum chances for a specific tree topology", *Systematic Zoology* 20 (1971), 406–419.

[12] W.M. FITCH and J.S. FARRIS, "Evolutionary trees with minimum nucleotide replacements from amino acid sequences", *Journal of Molecular Evolution* 3 (1974), 263–278.

[13] L.R. FOULDS and R. L. GRAHAM, "The Steiner problem in phylogeny is NP-Complete", *Advances in Applied Mathematics* 3 (1982), 43–49.

[14] L.R. FOULDS and R.L. GRAHAM, "Unlikelihood that minimal phylogenie for a realistic biological study can be constructed in reasonable computational time", *Mathematical Biosciences* 60 (1982), 133–142.

[15] E. GAMMA, R. HELM, R. JOHNSON, and J. VLISSIDES, *Design patterns – Elements of reusable object oriented software*, Addison Wesley, 1994, Reading.

[16] P.A. GOLOBOFF, "Methods for faster parsimony analysis", *Cladistics* 9 (1996), 199–220.

[17] P. A. GOLOBOFF, personnal communication, 1997.

[18] N. MLADENOVIĆ and P. HANSEN, "Variable neighbourhood search", *Computers and Operations Research* 24 (1997), 1097–1100.

[19] W. HENNIG, *Phylogenetic systematics*, University of Illinois Press, 1966, Urbana.

[20] R.E. JOHNSON, "Components, frameworks, patterns", 1997 (in ftp://st.cs.uiuc.edu/~papers/frameworks/framework97.ps).

[21] D. PENNY, L.R. FOULDS, and M.D. HENDY, "Testing the theory of evolution by comparing phylogenetic trees constructed from five different protein sequences", *Nature* 247 (1982), 197–200.

[22] M. LUCKOW and R.A. PIMENTEL, "An empirical comparison of numerical Wagner computer programs", *Cladistics* 1 (1985), 47–66.

[23] N.I. PLATNICK, "An empirical comparison of microcomputer parsimony programs", *Cladistics* 3 (1987), 121–144.

[24] N.I. PLATNICK, "An empirical comparison of microcomputer parsimony programs, II", *Cladistics* 5 (1989), 145–161.

[25] J. RUMBAUGH, "OMT: the object model", *Journal of Object Oriented Programming* 7 (1995), 21–27.

[26] J. RUMBAUGH, M. BLAHA, W. PREMERLANI, F. EDDY, and W. LORENSEN, *Object oriented modeling and design*, Prentice-Hall, 1991, Englewood Cliffs.

[27] D.D. SANKOFF and P. ROUSSEAU, "Locating the vertices of a Steiner tree in arbitrary space", *Mathematical Programming* 5 (1975), 240–249.

[28] E. SOBER, "Parsimony, likelihood and the principle of the common cause", *Philosophy of Science* 54 (1987), 465–469.

[29] D.L. SWOFFORD and G. OLSEN, "Phylogeny reconstruction", *Molecular systematics* (D.M. Hillis and C. Moritz, eds.), Sinauer, 1990, Sunderland.

[30] E.O. WILEY, D. SIEGEL-CAUSEY, D.R. BROOKS, and V.A. FUNK, *The compleat cladist: A primer of phylogenetic procedures*, Special publication no. 19, University of Kansas, Museum of Natural History, 1991.