Automatically Produced Algorithms for the Generalized Minimum Spanning Tree Problem

Carlos Contreras-Bolton, Carlos Rey, Sergio Ramos-Cossio, Claudio Rodríguez, Felipe Gatica and Victor Parada

1DEI, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy
2Departamento de Ingeniería Informática, Universidad de Santiago de Chile, 3659 Avenida Ecuador, 9170124 Santiago, Chile

Correspondence should be addressed to Carlos Contreras-Bolton; carlos.contreras3@unibo.it

Received 2 December 2015; Revised 2 February 2016; Accepted 16 February 2016

Academic Editor: Frédéric Saubion

The generalized minimum spanning tree problem consists of finding a minimum cost spanning tree in an undirected graph for which the vertices are divided into clusters. Such spanning tree includes only one vertex from each cluster. Despite the diverse practical applications for this problem, the NP-hardness continues to be a computational challenge. Good quality solutions for some instances of the problem have been found by combining specific heuristics or by including them within a metaheuristic. However studied combinations correspond to a subset of all possible combinations. In this study a technique based on a genotype-phenotype genetic algorithm to automatically construct new algorithms for the problem, which contain combinations of heuristics, is presented. The produced algorithms are competitive in terms of the quality of the solution obtained. This emerges from the comparison of the performance with problem-specific heuristics and with metaheuristic approaches.

1. Introduction

Determining the minimum cost spanning tree in a graph is a problem with various applications in the world of operations planning and management. It is known as the minimum spanning tree problem (MSTP) and consists of finding a tree of minimum cost that spans all vertices of a graph, the resolution of which can be obtained in polynomial time [1, 2]. However, there are several extensions of the MSTP that are generally NP-hard and for which it is not possible to obtain a solution in polynomial time [3]. A generalized version of the MSTP is the generalized minimum spanning tree problem (GMSTP), which still represents an enormous challenge because it belongs to the NP-hard class [4]. It has recently received much attention not only because of its difficulty but also because of its diverse applications. Dror et al. [5] study the design of a minimum-length irrigation network in agricultural irrigation in desert environments, while Myung et al. [4] consider supporting the decision making on the location of public facilities, commercial offices, and distribution centers connected by links such as roads or telecommunications. In the same field, Golden et al. [6] designed backbones in communication networks.

The GMSTP consists of finding a minimum cost spanning tree in an undirected graph, the vertices of which are divided into clusters such that the spanning tree includes only one vertex from each cluster. Let $G = (V, E)$ be an undirected graph with $n$ nodes, and let $V_1, \ldots, V_m$ be a division of $V$ into $m$ subsets called clusters; that is, $V = V_1 \cup V_2 \cup \cdots \cup V_m$ and $V_i \cap V_k = \emptyset, \forall i, k \in I = \{1, \ldots, m\}, I \neq k$. The cost of an edge $e = (u, v) \in E$ is denoted by $c_{uv} \in \mathbb{R}$ and the cost of a tree is obtained by adding the individual costs of the edges that compose it. Thus, the GMSTP consists of finding a tree of minimal cost that spans exactly one vertex $u_k \in V_k$, $k \in I$. Different formulations using integer programming have been proposed in the literature [7, 8]. Particularly, a compact formulation in terms of the number of variables and constraints was proposed by Pop [9, 10] by considering four types of binary variables.

Because of its computational complexity, the GMSTP has been addressed using a variety of approaches. The different mathematical formulations proposed have been unable to
solve large problems because long computational time or large memory resources are required [11]. Other approaches to solving the GMSTP are constructive and improvement heuristics. The former provide the solution by adding edges step by step until a generalized minimum spanning tree is constructed, for which adaptations of the well-known polynomial algorithms for the MSTP are used [5, 12].

In search of new solutions for the most challenging GMSTP instances some metaheuristics have been implemented. GRASP, tabu search, and genetic algorithms show good performance with larger instances; however, none of them determines the optimal solution for all larger sizes known instances. Such instances with 229 to 783 nodes were proposed to test the performance of a model based on tabu search [13] in which a current solution is represented by an array of size equal to the number of clusters and the neighbor solutions are randomly generated. When comparing the tabu search model with two versions of genetic algorithms [6] the result obtained by Öncan et al. [13] achieved better or equal solutions. In addition, an adaptive version of GRASP that uses path-relinking and iterated local search improved some of the solution values, through other subsets. However, the best known value was not achieved by this approach [11].

Recently, Contreras-Bolton et al. [14] proposed an approach based on a multiprocessor genetic algorithm by considering two crossover and five mutation operators, three of which are local searches. That allowed MGA to be competitive with respect to the best algorithms present in the literature, in terms of both quality of the solution and computing time.

Different approaches known for GMSTP include various combinations of search heuristics. There are different ways to generate a neighbor solution: local search algorithms, different operators, or efficient well-known algorithms that solve only a subproblem. When properly combined these heuristics give rise to novel and effective problem-solving techniques. Indeed, the genetic algorithm presented by Golden et al. [6] uses a local search method as a mutation operator and a particular designed crossover. In addition, Hao and Chaouachi [15] presented a random greedy algorithm that includes different search techniques. On the other hand, Ferreira et al. [11] presented five different heuristics that when combined with path-relinking give rise to six different versions of GRASP [16]. Such approach suggests that to obtain good results for GMSTP a proper selection and combination of heuristics is necessary. However, the combinations explored so far seem to be an only subset of the many possibilities that can be examined; thus by studying the unexplored combinations new algorithms for the GMSTP may be generated.

The task of automatically selecting and combining simple heuristics to generate a generic heuristic to solve any instance of a given optimization problem is known as a hyperheuristic [17, 18]. A hyperheuristic performs a search in the heuristic space rather than searching in the problem's solution space. Some of the most widely used approaches to generating hyperheuristics are genetic programming [19, 20], genetic algorithms [21], and learning systems [22]. The hyperheuristic approach has been used to approach various optimization problems, such as packing [23], timetabling [24], scheduling [25], MAX-SAT [26], vertex coloring problems [27], and binary knapsack problem [28, 29].

In this paper, an automated technique is presented that explores new heuristic combinations for the GMSTP. The algorithms are constructed from elementary heuristic components obtained from the methods described in the current literature and from a set of control structures typically used in any algorithm. The constructive process is carried out with a genetic algorithm in which a binary string represents the interactions between the elementary heuristics and the control structures [30–33].

In the following section, the procedures for generating the algorithms are described. The computational results of the generated algorithms are presented and discussed in the third section. The conclusions of the study are presented in the last section.

2. Procedure for Generating Algorithms

The procedure for generating the algorithms takes into account a genotype-phenotype genetic algorithm (GPGA) [34], a set of structures that composes the algorithms, and a fitness procedure to evaluate the performance. The GPGA considers two search spaces; the first search space corresponds to the genotype and is composed of strings which represent characteristics of the algorithms, and the second is the phenotype, composed of trees that assemble instructions that when executed find a solution to the GMSTP (Figure 1). Thus, from a population containing a fixed number of strings, a new population is generated by applying the selection, crossover, and mutation operators. In order to evaluate the performance of each string, the corresponding tree is constructed and evaluated. The process is repeated a number of times. The final algorithms are produced after two stages: First the algorithms are evolved by a number of generations until a convergence of the GPGA is detected and in a second stage, the best algorithms from the first stage are selected and evaluated with a different set of problem instances.

2.1. The Genotype for the GPGA. The genotype considered for the GPGA is a binary string encoding a combination of identifiers for both the heuristics and the control instructions. Each integer number represented in the binary string is coded by 5 bits thus representing 32 different components. Consequently, a binary string denotes the order in which the components are selected to construct the corresponding algorithm. In addition, a tree is constructed from the string using the preorder search [35]. Figure 1 describes a binary tree with 12 nodes constructed from a string of 60 bits.

2.2. The Phenotype for the GPGA. The phenotype is composed of trees that represent potential algorithms for the GMSTP. Such algorithms require a data structure definition and sets of functions and terminals to be considered in the algorithms as internal and leaf nodes, respectively.

2.2.1. Data Structure. The data structure organizes information to save both the graph and the results of the algorithms
produced. The Boost Graph Library [36] was used for the implementation of the graphs with two adjacent lists, one to save the problem instance and the other to contain the solution of the spanning tree. The STL library [37] was used to save the number of clusters and their associated vertices using maps and vectors.

2.2.2. Definition of Functions and Terminals. As elementary components of the algorithms to be produced, a set of functions and a set of terminals are defined. The former typically comprise any algorithm as the control structure. We used the following functions:

(i) \( \text{Equal}(P_1, P_2) \): \( P_1 \) and \( P_2 \) are executed. The function returns true if both arguments return the same Boolean value; otherwise, it returns false.

(ii) \( \text{And}(P_1, P_2) \): \( P_1 \) is executed; if it returns true, then \( P_2 \) is executed. If \( P_2 \) returns true, the function returns true. In any other case, the function returns false.

(iii) \( \text{Or}(P_1, P_2) \): \( P_1 \) is executed; if it returns true, the function returns true; otherwise, \( P_2 \) is executed. If \( P_2 \) returns true, the function returns true; otherwise, the function returns false.

(iv) \( \text{Not}(P_1) \): argument \( P_1 \) is executed. The function returns the logical negation of the result of the argument.

(v) \( \text{If}(P_1, P_2) \): argument \( P_1 \) is executed; if it returns true, argument \( P_2 \) is executed, and the function returns true. Otherwise, \( P_2 \) is not executed and the function returns false.

To prevent the formation of greedy algorithmic structures and also to achieve greater diversity in the produced algorithms, five control structures of the "while" type are proposed.

\( \text{While}(P_1, P_2) \) is described as follows: argument \( P_2 \) is executed if argument \( P_1 \) returns true. The first argument has an extra stop condition. Then, five varieties of \( \text{while} \) are modeled, as explained below:

(i) \( \text{While} \): if at most one-quarter of the clusters belong to the spanning tree, argument \( P_2 \) is executed.

(ii) \( \text{While2} \): if at most one-half of the clusters belong to the spanning tree, argument \( P_2 \) is executed.

(iii) \( \text{While3} \): if at most three-quarters of the clusters belong to the spanning tree, argument \( P_2 \) is executed.

(iv) \( \text{While4} \): if there is at least one cluster that does not belong to the spanning tree, argument \( P_2 \) is executed.

(v) \( \text{While5} \): it describes functions similar to the \( \text{for} \) instruction because argument \( P_2 \) is executed as many times as clusters exist in the problem.

Additionally, two extra stopping conditions are implemented. One initiates if more than 10 iterations occur that do not produce changes in the data structure. The other stop condition occurs if the number of iterations exceeds the number of vertices in the problem.

Each terminal is designed to return true if it executes the task for which the terminal is designed; otherwise, it returns false. Terminals can be grouped into three types: initial connection, construction, and improvement. The initial connection terminals are designed to operate only once, and their objective is to include the first vertex or edge into the solution. The initial connection terminals are described below:

(i) \( \text{prim-initial-connection} \): the vertex with the largest number of edges obtained by applying Prim’s elemental algorithm to the MSTP is added to the solution.

(ii) \( \text{least-vertex-initial-connection} \): the vertex with the least average cost of its edges is added to the solution.
(iii) **least-cluster-initial-connection**: the vertex connected to the lowest cost edge from the cluster that contains the smallest number of vertices is added.

(iv) **least-edge-initial-connection**: the two initial vertices that are part of the lowest cost edge are added.

The construction terminals add a vertex to the partial solution in each step:

(i) **connect-smallest-edge-with-the-tree**: the vertex with the smallest connection edge is added to the partial solution.

(ii) **connect-cluster-with-more-vertices**: the vertex that has the lowest cost edge of the cluster that contains the largest number of vertices is added to the partial solution.

(iii) **connect-cluster-with-fewer-vertices**: the vertex that has the lowest cost edge of the cluster that contains the smallest number of vertices is added to the partial solution.

The improvement terminals operate on the current solution:

(i) **subtree-k-cluster-connection-improvement**: this improvement is based on the heuristic of [38]. It improves the current partial solution by searching for *k* clusters that form a subtree of the partial solution. This subtree is composed of a leaf cluster of degree 1, a root cluster that can have a degree equal to or greater than 2, and other clusters with a degree of 2. All possible combinations of the *k − 1* clusters are formed, and the remaining cluster (tree root) remains fixed at its position and vertex. For this terminal, *k = \{2, 3, 4\}* is used. Furthermore, all clusters of the partial solution are exposed to this terminal so the improvement action can be applied as many times as possible.

(ii) **connection-cluster-improvement**: the current partial solution is improved by switching the current vertex of a cluster with another, producing a lower total cost. This procedure is applied to each cluster of the current solution.

(iii) **tree-leaf-connection-improvement**: the current partial solution is improved by connecting one leaf vertex to a different vertex of the tree, provided that the cost of the reconnected solution has a lower cost than the current cost. This procedure is followed for all edges of all leaf clusters.

(iv) **internal-edge-connection-improvement**: the current partial solution is improved by reconnecting two subgraphs generated by removing an internal edge of the current tree. The edge joining two subgraphs at minimum cost is chosen for the process. This procedure is repeated at all internal edges.

If the terminals succeed in decreasing the cost of the partial solution, it returns true; otherwise, it returns false.

### 2.3. The Fitness Procedure

The performance of an algorithm is evaluated by running it with a set of GMSTP instances. For this purpose a fitness function is defined and a set of test instances are selected, a portion of which are used in the first stage and the remainder is considered for evaluation of the selected algorithms. A fitness function *f* with two components is considered: the relative error and the relative deviation of the number of spanning tree edges. The relative error is measured with respect to the best solution known for each problem instance. Because the algorithms may generate solutions with fewer edges than the required number, the second component considers the relative difference with respect to \(m − 1\). Let \(I = \{1, 2, \ldots, \varphi\}\) be a set of instances of the GMSTP and let \(G_i^*\) be the optimum cost associated with instance *i*. In addition, \(y_i\) is the number of edges and \(G_i\) the cost of the tree obtained by the algorithm that solves the instance *i*. Additionally, we define \(\alpha\) and \(\beta\) as two real numbers in \([0, 1]\), where \(\alpha + \beta = 1.0\). Then, the fitness function is defined by

\[
f = \frac{1}{\varphi} \left( \alpha \sum_{i=1}^{\varphi} \frac{G_i^* - G_i}{G_i^*} + \beta \sum_{i=1}^{\varphi} \frac{m - y_i - 1}{m - 1} \right)
\]

Two groups of the problem instances listed in Table 1 were used [11]. The first group consists of 150 instances containing between 48 and 226 vertices with known optimum [39, 40]. The second group consisted of 101 instances containing between 229 and 783 vertices and was generated by Öncan et al. [13]; only the lower bounds for these instances are known. Each group was subdivided by type of clustering center; that is, \(\mu = 3\), \(\mu = 5\), \(\mu = 7\), and \(\mu = 10\), where \(\mu\) was the approximate mean number of vertices in a cluster [6]. For the evolution stage, the 29 instances of \(\mu = 3\) clustering type in group 1 were used. In the validation stage of the generated algorithms, all instances were used.

### 2.4. Operators and Parameters for GPGA

The appropriate selection of genetic algorithm operators has several options [41]. In this study, operators are selected based on previous studies dealing with the automatic generation of algorithms and preliminary runs [14, 27, 29]. Thus, a selection operator used double tournament which consists in selecting four individuals that compete first in fitness and then in size [42, 43]. In addition, the two-point crossover, the bit-flip mutation, and the generational population model were used [30].

The parameters used in GPGA were population size, number of generations, crossover and mutation probabilities, chromosome length, and \(\alpha\) and \(\beta\) parameters of the fitness.
function. The crossover probability was set to 0.9. The mutation probability was fine-tuned considering the following values: \{0.005, 0.01, 0.02, 0.04, 0.08, 0.10, 0.12, 0.14, 0.16, 0.18, 0.20\} and from this set, 0.12 was the best value in terms of the fitness function. Moreover, preliminary experiments that considered 50 strings by population showed that no improvements in the fitness function are detected after 200 generations suggesting this value as the stopping criterion for the GPGA. Furthermore, taking into account that the size of a generated algorithm affects its readability the chromosome’s length was set at 60 to obtain algorithms easy to be decoded. Additionally, \(\alpha = 0.6\) and \(\beta = 0.4\) are set to give more importance to the solution quality. Actually, the second term reaches a zero value during the first generations, ensuring that the algorithms only inspect feasible GMSTP solutions.

2.5. Hardware and Software Used. A cluster of computers with 2.10 GHz, 64 cores (32 were used), and 128 GB RAM with AMD Opteron™ 6272 Processors were used, with a GNU/Linux operating system (CentOS 6.2 distribution). The evolutionary algorithm was implemented in the C++ language, and the data structures were managed by the Boost Graph Library [36] and STL library [37]. The fitness function was evaluated in parallel using a set of processors with shared memory and the OpenMP library [44]. The evaluation cycle of each individual was thus asynchronous, and each processor evaluates one individual at a time. The mathematical model was implemented in C++ language and CPLEX 12.6 was run on a computer equipped with 2.10 GHz, 32 cores, and 64 GB RAM. The processor was an Intel(R) Xeon(R) CPU E7-4830. CPLEX runs were performed sequentially by using only one core.

3. Results and Discussion

The experiment consisted in executing the evolutionary algorithm 90 times with the set of grid instances \(\mu = 3\) of group 1. Table 2 displays the average, maximum, minimum, and the standard deviation values for the fitness, the height, and the number of nodes that the best algorithm found, including the number of generations in which each algorithm was found. The average time taken for the evolution was 9149.43 seconds. During the 90 evolutions, 6,892 algorithms were generated (on average), which satisfactorily solved the GMSTP. A total of 900,000 algorithms were generated in the entire process, of which 620,301 or 68.92\% were feasible algorithms, that is, algorithms that produced a feasible spanning tree. The average fitness is approximately 3.01\%, and the best algorithms had 1.9\% error in solving the 29 evolution instances. The size of the algorithm is between 16 and 39 nodes. It did not detect a fixed generation in which the best algorithm arises during the run; in fact, at least in one run, the better algorithm emerged in the initial population. The production process of the algorithms is expensive, requiring 2.5 hours on average for each run and as many as 41.1 hours for the longest run.

At the end of each run there is a convergence toward populations that contain individuals of variable quality, which have fitness values close to the best value of the population. Figure 2 indicates this typical behavior for the first 75 generations. The left ordinate axis gives the fitness value of the best algorithm in the population; the right ordinate gives the number of feasible algorithms; and the abscissa gives the generation number. In this case, the randomly generated population exhibits fitness values between approximately 0.048 and 1.00. Starting in generation 14, the average fitness approaches the value of the best individual, which is found in generation 43, marked in the figure as “minimum.” The results indicate that the fitness value of the worst individual fluctuates around the value of 1, with some occurrences near to the value of the best individuals. This behavior agrees with our definition of fitness; that is, individuals who do not cover all of the problem’s clusters have a penalized value. The presence of the worst individual is detected with less frequency because the number of feasible algorithms typically increases, following the trend illustrated in Figure 2; in consequence, many of the algorithms in a population are capable of finding feasible solutions for all 29 instances.

The different algorithms generated during the evolutionary process can be decoded manually from their tree structure to obtain their corresponding pseudocode. An illustrative case is presented in Algorithm 1 that builds a solution in

<table>
<thead>
<tr>
<th>Fitness</th>
<th>Height</th>
<th>Number of nodes</th>
<th>Generation found</th>
<th>Time (sec)</th>
<th>Algorithms generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>0.0301</td>
<td>5.79</td>
<td>22.13</td>
<td>69.60</td>
<td>6892.23</td>
</tr>
<tr>
<td>Min.</td>
<td>0.0188</td>
<td>4.00</td>
<td>16.00</td>
<td>0.00</td>
<td>3.00</td>
</tr>
<tr>
<td>Max.</td>
<td>0.1465</td>
<td>10.00</td>
<td>39.00</td>
<td>198.00</td>
<td>9573.00</td>
</tr>
<tr>
<td>St. dev.</td>
<td>0.0134</td>
<td>1.23</td>
<td>3.78</td>
<td>67.28</td>
<td>3512.81</td>
</tr>
</tbody>
</table>

Figure 2: Evolution of fitness and number of feasible algorithms.
two stages. The first stage operates in a constructive way, building a spanning tree that connects the minimum cost edges of the clusters that have the fewest vertices. This stage ends when it connects one-quarter of the total clusters. Then, it continues connecting in the same way, including new edges in the tree, applying improvements that consist in reconfiguring the current structure, and replacing higher cost edges with others of lower cost. Such operations are performed with the \textit{internal-edge-connection-improvement} terminal. To a lesser degree, the terminals \textit{connection-cluster-improvement}, \textit{tree-leaf-connection-improvement}, and \textit{subtree-4-cluster-connection-improvement} also act in the corresponding participation order. In general, the various algorithms that have been found contain common structures that have greedy and constructive characteristics, as observed in the algorithms of Kruskal and Prim. However, the most sophisticated algorithms have a constructive stage and an additional stage in which the construction and reconfiguration of the spanning tree are intercalated, similar to the stage in Algorithm 1. The intercalation primarily occurs due to the various \textit{while} functions.

The height of the algorithms tends to stabilize at approximately 4, which is the minimum possible value that a tree can have, according to the length of the chromosome. The initial randomly generated population determines a dispersed range of possible heights for the algorithms. Gradually, as the evolutionary process occurs, the range tends to decrease. A typical case is displayed in Figure 3, which corresponds to the run that generates Algorithm 1. Figure 3 depicts the values of the largest, average, and smallest heights for each population.
In the first population, algorithms with the minimum height are found, but starting in generation 13, those individuals disappear, and individuals are produced that are stabilized at a value of 5 until the end of the process. This behavior is caused by the double tournament selection genetic operator, which selects by fitness and by height. Thus, individuals with heights that oscillate between 5 and 8 can also be observed because of the probabilistic nature of the operator.

The algorithms obtained are numerically competitive with the existing heuristics for the GMSTP. The generated algorithms, similar to the existing heuristics, exhibit errors of small magnitude with respect to the optimum. In Table 3 the numerical results for three algorithms are displayed: GMSTP1, GMSTP2, and GMSTP3 that are produced and tested with instances of group 1. In addition, the solutions obtained by solving the local-global mixed integer programming formulation proposed by Pop [9, 10] are presented. Solutions are obtained with CPLEX denoting as CPLE1,1 runs with a running time limited to 300 seconds and CPLEX,2 runs with a running time limited to 3600 seconds. Table 3 also contains the results from three adaptations of the classical algorithms of Kruskal, Prim, and Sollin proposed by Feremans et al. [39], a simple heuristic called the spanning tree lower bound (STLB), three improved heuristics from the adaptations of Kruskal, Prim, and Sollin, IKH, IPH, and ISH, proposed by Golden et al. [6], and five heuristics (C1, RC2, RC3, RC4, and RC5) proposed by Ferreira et al. [11]. The first column of Table 3 displays the name of the heuristic. The following five columns correspond to the performance of each heuristic (percentage of error) and to the computational time (seconds) used for the subsets of the instances of group 1, grouped by type of clustering. The last column displays the performance for all instances of group 1. The sign ‘—’ means that the computational time was not available. In the results for the instances of group 1, GMSTP3, the best overall performance, is exhibited, with an average error of 5.42%, followed by GMSTP2 with 5.55% error; the closest to the other heuristics is ISH, with an error of 6.27%, followed by IKH and GMSTP1, with errors of 6.34% and 6.42%, respectively. In terms of the performance by clustering type, GMSTP2 exhibits the results with the lowest errors, that is, 3.28%, 4.90%, and 6.38% for $\mu = 3$, $\mu = 5$, and $\mu = 7$, respectively. However, regarding the clustering center, GMSTP3 exhibits a better performance, that is, an error of 4.71%. IKH exhibits the best performance for $\mu = 10$, with 6.76%, and GMSTP3 and GMSTP2 exhibit errors of 7.47% and 7.86%, respectively. With respect to time, GMSTP2 and GMSTP3 used average computational times of 0.029 and 0.027 seconds, respectively, to solve each of the 150 instances; ISH solved it in 17.60 seconds.

Table 4 displays the numerical results comparing the group 2 instances with the heuristic of Ferreira et al. [11]. This group of instances was not available in the paper by Golden et al. [6]. The data indicate that the three algorithms perform better than the five heuristics in all clustering type subsets of these instances. In addition, in the overall column, the new algorithms exhibit better results, with average errors between 5.99% and 6.07%, while the best RC2 heuristic results in an error of 8.92%. Moreover, the majority of the instances solved with GMSTP1, GMSTP2, and GMSTP3 use less computational time. With respect to time, GMSTP2 used an average of 6.336 seconds to solve each of the 101 instances in group 2, compared to the 0.005 seconds used by RC2.

Produced heuristics perform better than the two runs of CPLEX in terms of quality of solutions and computational time (Tables 3 and 4). Some optimal solutions are found for small instances by CPLEX,1 and CPLEX,2 runs and the average gaps in both runs are greater than the gaps for the three selected algorithms. Exceptions are found for $\mu = 7$ and $\mu = 10$. However, this result is not remained for the global average. Furthermore, with group 2 instances the gap degraded even more reaching values up to 636.61%. This fact shows that integer programming based algorithms are not yet competitive with heuristics or metaheuristics.

Table 5 displays the overall results for both groups. GMSTP2 achieved an average error of 5.58%, followed by GMSTP3 and GMSTP1, with errors of 5.65% and 6.28%, respectively. Of the other five heuristics, RC2 exhibited the best performance, with an average error of 10.10%. With respect to computational time, GMSTP2 solved each of the 251 instances in an average of 2.567 seconds, compared to the 0.002 seconds used by RC2.

Although requiring low computational time, the produced algorithms do not obtain the same solution qualities as the tabu search, GRASP, and MGA. Table 6 shows the difference between the performances of the three new algorithms and the metaheuristics. The use of different software and hardware in the three studies produces an unfair comparison of the computer times in Table 6. On the contrary, we note a clear difference in the solution qualities. The produced algorithms execute a set of constructive and improvement instructions during a fixed number of steps; thus, a variation in computer time does not imply a change in the number of iterations. Tabu search, GRASP, and MGA repeat a metarule until a convergence gap is found, and thus, a longer computing time may achieve a better solution. These reasons partly explain the differences in performance. Therefore, more research is needed to integrate the strengths of these approaches.

4. Conclusions

This paper presents a technique to obtain new combinations of heuristics for the GMSTP. The algorithms were produced with an evolutionary computation procedure designed and implemented to combine components of elementary heuristics and control structures. The algorithms thus generated are characterized by a first stage that contains structures similar to those found in the heuristics that already exist for the problem. In the second stage, the algorithms implement several methods to improve the partial solution.

The algorithms were evolved using 90 executions on 29 instances chosen from group 1, which includes all of $\mu = 3$ clustering grid types. Then, a set of 251 instances was used to test the algorithms. Including the new while functions helped the generated algorithms to work in stages and avoid becoming structured as a single large constructive stage.
Table 3: Numerical results using group 1 instances.

<table>
<thead>
<tr>
<th>Clustering type</th>
<th>Center</th>
<th>$\mu = 3$</th>
<th>$\mu = 5$</th>
<th>$\mu = 7$</th>
<th>$\mu = 10$</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gap (%)</td>
<td>Time (sec)</td>
<td>Gap (%)</td>
<td>Time (sec)</td>
<td>Gap (%)</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>CPLEX_1</td>
<td>53.04</td>
<td>170.4870</td>
<td>214.98</td>
<td>204.4361</td>
<td>92.54</td>
<td>182.3282</td>
</tr>
<tr>
<td>CPLEX_2</td>
<td>10.33</td>
<td>1039.8365</td>
<td>130.72</td>
<td>1349.2061</td>
<td>19.56</td>
<td>1255.2257</td>
</tr>
<tr>
<td>STLb</td>
<td>37.13</td>
<td>—</td>
<td>22.22</td>
<td>—</td>
<td>32.95</td>
<td>—</td>
</tr>
<tr>
<td>Kruskal</td>
<td>8.27</td>
<td>—</td>
<td>5.05</td>
<td>—</td>
<td>10.65</td>
<td>—</td>
</tr>
<tr>
<td>Prim</td>
<td>13.10</td>
<td>—</td>
<td>7.94</td>
<td>—</td>
<td>12.41</td>
<td>—</td>
</tr>
<tr>
<td>SolIn</td>
<td>7.99</td>
<td>—</td>
<td>5.22</td>
<td>—</td>
<td>10.70</td>
<td>—</td>
</tr>
<tr>
<td>IKH</td>
<td>4.80</td>
<td>—</td>
<td>3.42</td>
<td>—</td>
<td>6.48</td>
<td>—</td>
</tr>
<tr>
<td>IPH</td>
<td>6.81</td>
<td>—</td>
<td>5.06</td>
<td>—</td>
<td>6.45</td>
<td>—</td>
</tr>
<tr>
<td>ISH</td>
<td>4.82</td>
<td>—</td>
<td>3.60</td>
<td>—</td>
<td>6.59</td>
<td>—</td>
</tr>
<tr>
<td>C1</td>
<td>27.52</td>
<td>0.0001</td>
<td>16.21</td>
<td>0.0004</td>
<td>28.70</td>
<td>0.0003</td>
</tr>
<tr>
<td>RC2</td>
<td>8.40</td>
<td>0.0003</td>
<td>5.28</td>
<td>0.0003</td>
<td>10.86</td>
<td>0.0003</td>
</tr>
<tr>
<td>RC3</td>
<td>11.48</td>
<td>0.0005</td>
<td>7.22</td>
<td>0.0003</td>
<td>13.45</td>
<td>0.0001</td>
</tr>
<tr>
<td>RC4</td>
<td>16.07</td>
<td>0.0002</td>
<td>10.05</td>
<td>0.0013</td>
<td>15.24</td>
<td>0.0003</td>
</tr>
<tr>
<td>RC5</td>
<td>11.87</td>
<td>0.0120</td>
<td>7.80</td>
<td>0.0600</td>
<td>11.37</td>
<td>0.0171</td>
</tr>
<tr>
<td>GMSTP1</td>
<td>6.04</td>
<td>0.0134</td>
<td>3.52</td>
<td>0.0618</td>
<td>5.89</td>
<td>0.0208</td>
</tr>
<tr>
<td>GMSTP2</td>
<td>5.34</td>
<td>0.0128</td>
<td>3.28</td>
<td>0.0985</td>
<td>4.90</td>
<td>0.0209</td>
</tr>
<tr>
<td>GMSTP3</td>
<td>4.71</td>
<td>0.0185</td>
<td>3.39</td>
<td>0.0847</td>
<td>5.01</td>
<td>0.0182</td>
</tr>
</tbody>
</table>

**Overall**
- Avg. gap (%)
- Avg. time (sec)
Table 4: Numerical results using group 2 instances.

| Clustering type | Center | $\mu = 3$ |  | $\mu = 5$ |  | $\mu = 7$ |  | $\mu = 10$ |  | Overall |
|-----------------|--------|-----------|---------------|-----------|---------------|-----------|---------------|-----------|----------|
|                 | Gap (%)| Time (sec) | Gap (%)       | Time (sec) | Gap (%)       | Time (sec) | Gap (%)       | Time (sec) | Avg. gap (%)| Avg. time (sec) |
| CPLEX,1         | 796.47 | 300.00    | 406.34        | 300.00    | 747.82        | 300.00    | 684.87        | 300.00    | 636.61    | 300.00 |
| CPLEX,2         | 738.79 | 3600.00   | 406.34        | 3600.00   | 710.08        | 3600.00   | 622.84        | 3600.00   | 576.00    | 3571.97 |
| C1              | 20.51  | 0.0027    | 14.41         | 0.1640    | 22.64         | 0.0028    | 27.60         | 0.2423    | 36.19     | 0.0000 |
| RC2             | 7.06   | 0.0038    | 4.94          | 0.0104    | 8.47          | 0.0046    | 10.59         | 0.892     | 13.65     | 0.0018 |
| RC3             | 8.89   | 0.0051    | 6.53          | 0.0126    | 10.53         | 0.0052    | 12.07         | 1.505     | 14.53     | 0.0022 |
| RC4             | 17.67  | 0.0034    | 12.68         | 0.0092    | 18.69         | 0.0036    | 22.25         | 19.94     | 28.53     | 0.0008 |
| RC5             | 11.30  | 0.8094    | 8.08          | 3.0652    | 11.85         | 0.8861    | 14.79         | 13.35     | 18.62     | 1.1653 |
| GMSTP1          | 5.23   | 20.41     | 3.56          | 127.251   | 6.02          | 25.478    | 7.06          | 6.07      | 8.52      | 0.4677 |
| GMSTP2          | 5.06   | 28.051    | 3.30          | 236.258   | 5.55          | 35.529    | 5.96          | 5.62      | 8.26      | 0.5968 |
| GMSTP3          | 5.09   | 26.379    | 3.33          | 190.505   | 5.94          | 29.034    | 6.91          | 5.99      | 8.71      | 0.4374 |

Scientific Programming 9
Table 5: Summary of the heuristics using group 1 and group 2.

<table>
<thead>
<tr>
<th>Heuristics</th>
<th>Avg. gap (%)</th>
<th>Avg. time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>29.05</td>
<td>0.0015</td>
</tr>
<tr>
<td>RC2</td>
<td>10.11</td>
<td>0.0020</td>
</tr>
<tr>
<td>RC3</td>
<td>12.22</td>
<td>0.0025</td>
</tr>
<tr>
<td>RC4</td>
<td>17.83</td>
<td>0.0018</td>
</tr>
<tr>
<td>RC5</td>
<td>12.59</td>
<td>0.4807</td>
</tr>
<tr>
<td>GMSTP1</td>
<td>6.28</td>
<td>1.5124</td>
</tr>
<tr>
<td>GMSTP2</td>
<td>5.58</td>
<td>2.5670</td>
</tr>
<tr>
<td>GMSTP3</td>
<td>5.65</td>
<td>2.1015</td>
</tr>
</tbody>
</table>

Table 6: Summary of the heuristics and metaheuristics using group 1 and group 2.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Group 1 Avg. gap (%)</th>
<th>Group 2 Avg. gap (%)</th>
<th>Overall Avg. gap (%)</th>
<th>Avg. time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMSTP1</td>
<td>6.42</td>
<td>6.07</td>
<td>6.28</td>
<td>1.5124</td>
</tr>
<tr>
<td>GMSTP2</td>
<td>5.55</td>
<td>5.62</td>
<td>5.58</td>
<td>2.5670</td>
</tr>
<tr>
<td>GMSTP3</td>
<td>5.42</td>
<td>5.99</td>
<td>5.65</td>
<td>2.1015</td>
</tr>
<tr>
<td>TS</td>
<td>0.01</td>
<td>0.03</td>
<td>0.02</td>
<td>736.8500</td>
</tr>
<tr>
<td>GRASP</td>
<td>0.00</td>
<td>0.01</td>
<td>0.01</td>
<td>59.1300</td>
</tr>
<tr>
<td>MGA</td>
<td>—</td>
<td>0.01</td>
<td>—</td>
<td>49.43</td>
</tr>
</tbody>
</table>

In general, the height of an algorithm converges toward the minimum possible value. The new algorithms were competitive in terms of the quality of the solution obtained compared to the existing heuristics, and the errors for the 251 instances were smaller than the errors obtained using the heuristics considered.

**Conflict of Interests**

The authors declare that there is no conflict of interests regarding the publication of this paper.

**Acknowledgment**

The authors would like to thank the Complex Engineering Systems Institute (ICM: P-05-004-F, CONICYT: FBO16, DICYT: 61219-USACH, ECOS/CONICYT: CIJ3E04, and STI-CAMSUD: 13STIC-05).

**References**


