Production
ISSN: 0103-6513
production@editoracubo.com.br
Associação Brasileira de Engenharia de Produção
Brasil

SOARES VIANNA, DALESSANDRO; CLAUDIO ARROYO, JOSÉ ELIAS; SAMPAIO VIEIRA, PEDRO; RIBEIRO DE AZEREDO, THIAGO
Parallel strategies for a multi-criteria GRASP algorithm
Production, vol. 17, núm. 1, enero-abril, 2007, pp. 84-93
Associação Brasileira de Engenharia de Produção
São Paulo, Brasil

PRODUÇÃO

Parallel strategies for a multi-criteria GRASP algorithm

Dalessandro Soares Vianna
José Elias Claudio Arroyo
Pedro Sampaio Vieira
Thiago Ribeiro de Azeredo
UCAM-Campos

Abstract

This paper proposes different strategies of parallelizing a multi-criteria GRASP (Greedy Randomized Adaptive Search Problem) algorithm. The parallel GRASP algorithm is applied to the multi-criteria minimum spanning tree problem, which is NP-hard. In this problem, a vector of costs is defined for each edge of the graph and the goal is to find all the efficient or Pareto optimal spanning trees (Pareto-optimal solutions). Each process finds a subset of efficient solutions. These subsets are joined using different strategies to obtain the final set of efficient solutions.

The multi-criteria GRASP algorithm with the different parallel strategies are tested on complete graphs with n = 20, 30 and 50 nodes and r = 2 and 3 criteria. The computational results show that the proposed parallel algorithms reduce the execution time and the results obtained by the sequential version were improved.

Key words

Parallel GRASP algorithm, multi-criteria combinatorial optimization, minimum spanning tree.

Estratégias de paralelização para um algoritmo GRASP multicritério

Resumo

Este artigo propõe diferentes estratégias de paralelização de um algoritmo GRASP (*Greedy Randomized Adaptive Search Procedure*) multicritério. O algoritmo paralelo proposto é aplicado ao problema da árvore geradora mínima multicritério, que é NP-difícil. Neste problema, um vetor de custos é definido para cada aresta do grafo e o objetivo é encontrar as árvores geradoras eficientes (soluções Pareto-ótimas). Cada processo encontra um subconjunto de soluções eficientes. Estes subconjuntos são reunidos usando diferentes estratégias para obter o conjunto final de soluções eficientes.

O algoritmo proposto é testado em grafos completos com $n = 20$, 30 e 50 nós e $r = 2$ e 3 critérios. Os resultados computacionais mostram que a proposta de se paralelizar o algoritmo reduz o tempo de execução e os resultados obtidos pela versão seqüencial foram melhorados.

Palavras-chave

Algoritmo GRASP paralelo, otimização combinatória multicritério, árvore geradora mínima.

INTRODUCTION

Many practical optimization problems, generally, involve the minimization (or maximization) of several conflicting decision criteria. For example, in the topological network design problem is desirable to find the best layout of components optimizing performance criteria, such as financial cost, message delay, traffic, link reliability, and so on. These criteria are conflicting and can not be optimized simultaneously. Instead, a satisfactory trade-off has to be found. So a Decision Maker has to select the best compromise solution, taking into account the preference of the criteria.

The goal of multi-criteria combinatorial optimization (MCCO) is to optimize simultaneously $r > 1$ criteria or objectives finding a satisfactory trade-off. MCCO problems have a set of optimal solutions (instead of a single optimum) in the sense that no other solutions are superior to them when all criteria are taken into account. They are known as *Pareto optimal* or *efficient* solutions.

Solving MCCO problems is quite different from single-objective case ($r = 1$), where an optimal solution is searched. The difficulty is not only due to the combinatorial complexity as in single-objective case, but also to the research of all elements of the efficient set, whose cardinality grows with the number of objectives.

In the literature, some authors have proposed exact methods for solving specific MCCO problems (EHRGOTT; GANDIBLEUX, 2000). These methods are generally valid for two criteria ($r = 2$) problems, but can not be adapted easily to a higher number of criteria. Also, the exact methods are inefficient to solve large-scale NP-hard MCCO problems. As in the single-criterion case, the use of heuristic/metaheuristic techniques seems to be the most promising approach to MCCO because of their efficiency, generality and relative simplicity of implementation. Hard problems require large search spaces resulting in high computational costs. In this context, metaheuristics may require a large amount of time to find good feasible solutions, encouraging the use of parallel techniques (DRUMMOND *et al.*, 2001; OCHI *et al.*, 1999; VIANNA *et al.*, 1999). Although the main goal of a parallel metaheuristic is the reduction of the execution time necessary to find an acceptable solution, sometimes it can also be used to improve the results obtained by sequential versions.

In a recent overview of multi-criteria metaheuristics, Jones *et al.* (2002) report the increase of papers published in the nineties and also note that almost 80% of the papers are dedicated to real problems, especially in the discipline of engineering. This number reflects not only the increasing awareness of problems with multiple criteria, but also that metaheuristics are effective techniques to cope with such problems. Metaheuristics such as genetic algorithms (Holland, 1975), tabu search (Glover & Laguna, 1997) and simulated annealing (KIRKPATRICK *et al.*, 1983) were originally conceived for single-criterion combinatorial optimization and the success achieved in their application to a very large number of problems has stimulated researchers to extend them to MCCO problems. In (JONES *et al.*, 2002) is commented that there is no sign in the literature reviewed of the newer metaheuristic techniques, such as GRASP, being applied in the multi-criteria case. More recently, the first papers using the GRASP metaheuristic for multi-criteria problems were proposed (ARROYO *et al.*, 2005; VIANNA; ARROYO, 2004).

In the multi-criteria minimum spanning tree (mc-MST) problem, a vector of costs is defined for each edge of the graph and the problem is to find all Pareto optimal or efficient spanning trees (solutions). The literature on mc-MST problem is rather scarce. An exact method is proposed in (RAMOS *et al.*, 1998). In (EHRGOTT; KLAMROTH, 1997) and (HAMACHER; RUHE, 1994) are proposed approximate polynomial algorithms. The method proposed in (KNOWLES, 2002) and (ZHOU; GEN, 1999) are based on genetic algorithms.

In this paper, we propose three different ways of parallelizing a multi-criteria GRASP algorithm. The parallel GRASP algorithm is applied to the mc-MST problem with two and three criteria.

The organization of the paper is described as following. In the next section, we present the formulation of the mc-MST problem. In Section 3, we discuss with more details the **mc-GRASP** algorithm. The parallel GRASP strategies are presented in Section 4. We present computational results in Section 5. Finally, Section 6 contains our concluding remarks.

MULTI-CRITERIA MINIMUM SPANNING TREE PROBLEM

Let $G = (V, A)$ be a connected and undirected graph, where $V = \{v_1,...,v_n\}$ is a finite set of nodes and $A = \{e_1,...,e_m\}$ is a finite set of arcs or edges $ek = (i, j)$, $i \in V$, $j \in V$, $i \neq j$. Each edge $ek = (i, j)$ has associated a vector $cij = (cij^1,\ldots,cij^r)$ of r positive real numbers (costs). A spanning tree of graph G is a subgraph $T = (V, A_T)$ with $A_T \subseteq A$, such that T contains all nodes in V and connects them with exactly n–1 edges, so that there are no cycles. The mc-MST problem can be formulated as:

$$\text{Minimize } f(T) = (f_1(T),...,f_r(T))$$
$$\text{subject to } T \in \Omega,$$

where $f_k(T) = \sum_{\forall (i,j) \in A_T} c_{ij}^k$ is the k-th objective function and Ω

is the set of all the spanning trees of graph G. The image of a solution $T \in \Omega$ is the point $z = f(T)$ in the objective space $f(\Omega)$.

A point $z = f(T)$ *dominates* $z' = f(T')$ if $z_j = f_j(T) \leq z_j' = fj(T')$, $\forall j = 1,...,r$, and $z_j < z_j'$ for at least one j. A solution T dominates T' if $f(T)$ dominates $f(T')$. A solution $T^* \in \Omega$ is *efficient* (or *Pareto-optimal*) if there is no $T \in \Omega$ such that $f(T)$ dominates $f(T^*)$. The goal is to determine the set E of efficient solutions. We call the representation of set E in $f(\Omega)$ *Pareto-frontier*.

A utility function is a model of the Decision Maker's preferences that maps each point in the objective space into a value of utility. It assumed that the goal of the Decision Maker is to minimize the utility. In this paper is used the *weighted linear utility function* defined in the following way:

$$u(T) = \sum_{j=1}^{r} \lambda_j f_j(T),$$

where $\lambda = (\lambda_1,...,\lambda_r)$ is the weight vector such that

$$\sum_{j=1}^{r} \lambda_j = 1, \quad \lambda_j \geq 0, j = 1,...,r.$$

THE MULTI-CRITERIA GRASP HEURISTIC (MC-GRASP)

GRASP - Greedy Randomized Adaptive Search Procedure (Feo & Resende, 1995) is a multi-start metaheuristic, in which each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution using a greedy randomized algorithm, whose neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result.

The **mc-GRASP** heuristic is based on the optimization of the weighted linear utility function $u(T)$. The main idea of the heuristic is to define a weight vector for each iteration, which is used to calculate the function $u(T)$. At each iteration of the heuristic, a solution T is built using the greedy randomized Kruskal's algorithm (see Subsection 3.1). The Kruskal algorithm (KRUSKAL, 1956) is used replacing the vector of edges costs by a weighted linear combination of these costs. The built solution is submitted to a local search procedure (see Subsection 3.2).

The weight vector $\lambda = (\lambda_1,...,\lambda_r)$, generally, determinates a search direction on the Pareto-optimal frontier and various search directions are required to find a variety of Pareto optimal solutions. Murata *et al*. (2001) introduces a way of generating weight vectors distributed uniformly on the Pareto frontier. The vectors are generated combining r non-negatives integers with the sum of an integer value s:

$$w_1 + w_2 + ... + w_r = s, \text{ where } w_i \in \{0, 1,..., s\}$$

As an example, for $r = 2$ criteria and $s = 5$ we have 6 vectors: (5,0), (4,1), (3,2), (2,3), (1,4) and (0,5). For $r = 3$ and $s = 3$ we have 10 vectors: (3,0,0), (2,1,0), (2,0,1), (1,2,0), (1,1,1), (0,2,1), (0,3,0), (1,0,2), (0,1,2) and (0,0,3).

In order to obtain normalized weights $\sum_{j=1}^{r} \lambda_j = 1$, we considered $\lambda_j = w_j/s$, $w_j \in \{0,...,s\}$.

The number of generated vectors for r objectives and for a value of s, $N_r(s)$, is calculated as follows:

$$N_2(s) = s + 1.$$

$$N_3(s) = \sum_{i=0}^{s} N_2(i) = \sum_{i=0}^{s} (i+1) = (s+1)(s+2)/2.$$

$$N_4(s) = \sum_{i=0}^{s} N_3(i) = \sum_{i=0}^{s} (i+1)(i+2)/2.$$

Figure 1 presents the implemented **mc-GRASP** algorithm that receives as input parameters the number of iterations *N_iter*, the percentage $\alpha \in [0, 1]$ (controls the amount of greediness and randomness) used at the construction phase and the weighted utility function to be optimized. As output, the algorithm returns the *lPareto* list, where the nondominated solutions are stored. The number of iterations of the algorithm corresponds to the number of weight vectors.

Greedy randomized construction

In the construction algorithm (**Greedy_Randomized_Kruskal**), for each edge (i, j) of the graph is computed the weighted sum $\lambda c_{ij} = \sum_{k=1}^{r} \lambda_k c_{ij}^k$, where $c_{ij} = (c_{ij}^1,..., c_{ij}^r)$ is the cost vector of the edge (i, j) and $\lambda = (\lambda_1,...,\lambda_r)$ is the weight vector.

The candidate list C contains all the edges, in a no decreasing order of λc_{ij} ($C = \{e_1, ...,e_m\}$). The restricted candidate list is defined as $RCL = \{e_1,...,e_{|RCL|}\}$, where $|RLC| = \mathbf{max}(1, \alpha \times |C|)$ is the cardinality of RLC and $\alpha \in [0,1]$. At each iteration of the constructive phase, an edge is selected randomly from RCL and it is added to the partial spanning tree as in the Kruskal's algorithm (KRUSKAL, 1956). This phase finalizes when the spanning tree has n–1 edges. Note that, for $\alpha = 0$ we have the original Kruskal's algorithm. In the construction algorithm, randomization is necessary ($\alpha > 0$) to build different initial solutions.

Figure 2 presents the **Greedy_Randomized_Kruskal** algorithm that receives as input the parameter $\alpha \in [0, 1]$ (that controls the amounts of greediness and randomness) and the weight vector λ. As output, the algorithm returns the constructed tree *T*.

Local search

In the local search, a feasible spanning tree *T* is represented by a Prufer number *P* (vector with *n*–2 nodes) (Moon, 1967) and by a permutation of the *n* nodes *B*. *P* and *B* are constructed using the Encode algorithm of Figure 3.

Figure 1: mc-GRASP algorithm.

Algorithm mc-GRASP (*N_iter,* α*, u*)
01. *lPareto* $\leftarrow \varnothing$;
02. Define a set of weight vectors $\Lambda = \{\lambda^i = (\lambda_1,...,\lambda_r); i = 1,\ldots,N\_iter\}$;
03. **For** $i \leftarrow 1$ to *N_iter* **do**
04. **Begin**
05. $T \leftarrow$ **Greedy_Randomized_Kruskal** (α, λ^i);
06. **Update_The_Pareto_List** (*T, lPareto*);
07. **Local_Search** (T, λ^i, $u(T)$, *lPareto*);
08. **End_for**
09. **Return** *lPareto*;
End-Algorithm

Figure 2: Greedy_Randomized_Kruskal algorithm.

Algorithm Greedy_Randomized_Kruskal (α , λ)
01. Build the candidate list C with all edges c_{ij} of the graph, in a no decreasing order of $\lambda c_{ij} = \sum_{k=1}^{r} \lambda_k c_{ij}^k$;
02. $T \leftarrow \varnothing$;
03. $z \leftarrow 0$;
04. **While** $z < n - 1$ **do**
05. **Begin**
06. The restricted candidate list (RCL) is defined by the firsts h edges of C, where h = **max** (1, α x |C|) is the size of *RLC*;
07. Select edge e at random from *RCL*;
08. **If** T $\cup$ {e} does not create a cycle **then**
09. **Begin**
10. $T \leftarrow T \cup \{e\}$;
11. z $\leftarrow$ z + 1;
12. End_if
13. End_while
14. Return *T*;
End-Algorithm

The **Local_Search** procedure, showed in Figure 5, is based on the exclusion and addition of edges. Figure 4(a) shows an example of a tree T represented by B = [1 3 4 2 6 5 7] and P = [6 7 2 6 5] (B and P were constructed using the algorithm **Encode** of Figure 3). A neighbor T' of T (Figure 4(a)) is showed in Figure 4(b). The excluded edge is ($B[j]$, $P[j]$) = (6, 5), j = 5. This exclusion creates two sub-trees T_1 and T_2, rooted at nodes 6 and 5, respectively. T' is generated adding edge $(6, k) = (6, 7)$, $k \in T_2$, $k \neq 5$. The neighbor tree T' is represented by B' = [1 4 2 6 3 5 7] and P' = [6 2 6 7 7].

PARALLEL GRASP STRATEGIES

Three different ways of parallelizing a multi-criteria GRASP algorithm were implemented for the **mc-GRASP** algorithm, described in Section 3. In these algorithms, the set of weight vectors λ is divided among the p processes and each one executes N_iter/p GRASP iterations, where N_iter is the total number of iterations to be executed. Each process i ($1 \leq i \leq p$) works with its own local Pareto list (*lParetoi*). These lists are joined to generate the global Pareto list *lPareto*. Each algorithm has a different way of joining the local Pareto lists.

mc-ParGRASP1

In this algorithm, the local Pareto lists are joined only at the end of the algorithm. Each process i ($1 \leq i \leq p$) executes all the N_iter/p iterations and, at the end, sends the obtained list *lParetoi* to the master process (process 1).

This strategy has a simple implementation, but it has the inconvenience of all processes j ($2 \leq j \leq p$) stay idle while the master process receives the local lists and joins them into the global Pareto list *lPareto*. Depending on the number of solutions in each local Pareto list, this join procedure can consume a considerable time.

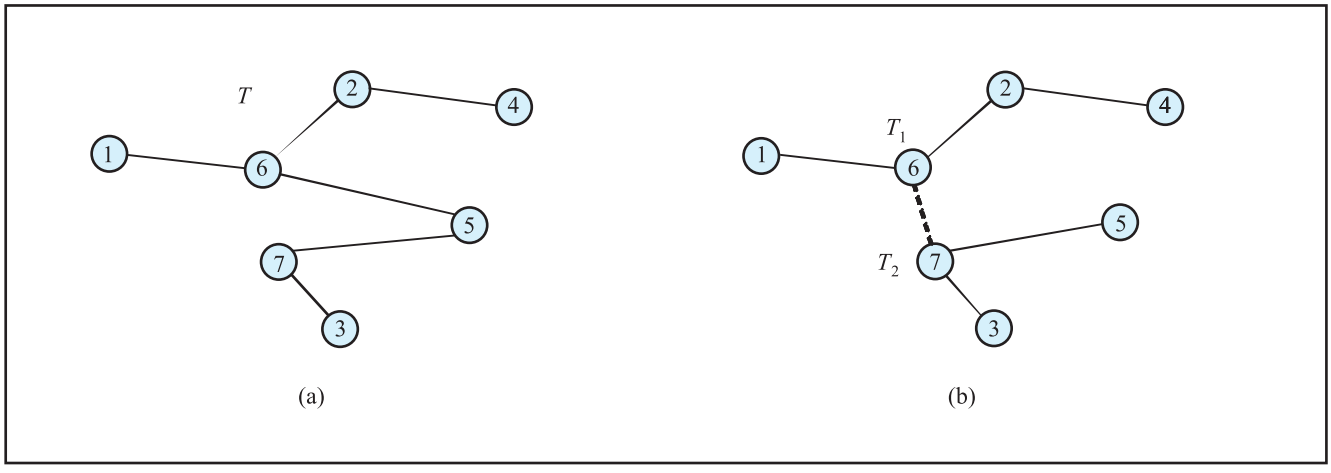
mc-ParGRASP2

In this algorithm, each process i ($2 \leq i \leq p$) sends, at each

Figure 3: Encode algorithm.

Algorithm Encode (T)
01. $P \leftarrow \varnothing$, $B \leftarrow |\varnothing|$ All n-1 edges in the tree T are labeled as temporary.
02. Construct a set $D_1 \subset V$ formed by degree 1 nodes of the temporarily labeled edges of T.
03. Choose node k, such that k is the least index in D_1.
04. Consider edge $(k, j) \in T$. $B \leftarrow B \cup \{k\}$ and $P \leftarrow P \cup \{j\}$.
05. Give edge $(k, j) \in T$ a permanent label. If there is only a remaining edge (u, v) with temporary label, add u and v to B, return P and B, stop. Else, go to Step 2.
End-Algorithm

Figure 4: (a) An example of an encoding, (b) drop-and-add neighbor.



z iteration, the new nondominated solutions (new solutions of $lPareto_i$) to the master process (process 1).

The goal of this implementation is to reduce the idleness time of the processes. After sending the solutions to the master process at iteration y, the process i ($2 \le i \le p$) can start the iteration y+1, while the master process updates the global Pareto list *lPareto*.

This strategy has an advantage: it consumes more communication time. A greater number of solutions is sent when compared with the **mc-ParGRASP1** algorithm.

mc-ParGRASP3

In this algorithm, at the moment process i ($2 \le i \le p$) finds a nondominated solution T, T is sent to process i-1. At the

Figure 5: Local_Search algorithm.

Algorithm Local_Search (T, λ, $u(T)$, *lPareto*)
01. **Encode**(T); //it is determined the Prufer number P and the B vector
02. *Improved* = **True**;
03. **While** *Improved* **do**
04. **Begin**
05. $u(T^*) = u(T)$;
06. **For** $j = 1$ **to** n–2 **do**
07. **Begin**
08. Delete edge $e = (B[j], P[j]) \in T$ creating two sub-trees T_1 and T_2, rooted at $B[j]$, and $P[j]$, respectively.
09. **For** each $k \in T_2$, $k \neq P[j]$ **do**
10. **Begin**
11. Construct a new tree T' adding an edge $e' = (B[j], k)$.
12. Calculate $u(T') = \sum_{j=1}^{r} \lambda_j f_j(T)$, where $fi(T') = f_i(T) - c_e^i + c_{e'}^i$, $i = 1,\ldots,r$.
13. **If** $u(T') < u(T^*)$ **then**
14. **Begin**
15. $u(T^*) = u(T')$;
16. Construct the representations B^* and P^* of the tree T' doing two passes trough B. In the first step, all the nodes $B[l] \in T_1 \cap B - \{B[j]\}$ are added to B^* and the correspondent adjacent nodes $P[l] \in P$ to P^*. Next, the nodes in $(B[j], k)$ are added to B^* and P^*, respectively. Finally all the nodes $B[l] \in T_2 \cap B$ are added to B^* (and the corresponding nodes $P[l] \in P$ to P^*, $l \le n$–2).
17. **End_if**;
18. **End_for**;
19. $T = T \cup \{e\}$;
20. **End_for**;
21. **If** $u(T^*) < u(T)$ **then**
22. **Begin**
23. $T = T^*$; $B = B^*$; $P = P^*$;
24. **Update_The_Pareto_List** (T, *lPareto*);
25. **Else**
26. *Improved* = **False**;
27. **End_if**
28. **End_while**;
29. **Return** *lPareto*;
30. **End Local_Search**

moment process i-1 receives the solution T, it verifies if T is a nondominated solution comparing it with the solutions in $lPareto_{i-1}$. If T is a nondominated solution, it is inserted into $lPareto_{i-1}$ and it is sent to the process i-2. This procedure is repeated until T is received by the master process (process 1) or it is not a nondominated solution in $lPareto_j$ ($1 \leq j \leq p$). In this way, the $lPareto_{j-1}$ is always more updated than $lPareto_j$, for $2 \leq j \leq p$.

The results show that parallelizing a multi-criteria GRASP algorithm reduces execution time and can also improve the set of nondominated solutions obtained by the sequential version.

The goal of this implementation is to decentralize the updating of the global Pareto list ($lPareto$).

This strategy has the same advantage of the previous one. It consumes more communication time, sending a greater number of solutions when compared with the **mc-ParGRASP1** algorithm.

COMPUTATIONAL EXPERIMENTS

The proposed parallel algorithms were implemented using the C programming language and MPI library for the parallelism.

The computational experiments were carried out in a SUN FIRE 6800 with SPARC III 750MHZ processors and 24Gb RAM.

The proposed parallel algorithms are tested on complete graphs with n = 20, 30 and 50 nodes and r = 2 and 3 criteria.

In the experiments done, N_iter = 5000 GRASP iterations were executed. The values of α, used during the construction phase in the **mc-GRASP** algorithm, are α = 0.08, 0.03 and 0.01 for graphs with n = 20, 30 and 50 nodes, respectively.

In the first experiment, the proposed parallel algorithms were executed with p = 4 processors. The interval to send nondominated solutions by the **mc-ParGRASP2** algorithm was z = 1 iteration. Tables 1 and 2 present, for each instance, the number n of nodes, the number r of criteria and, for each algorithm, the consumed time in seconds (t) and the total number of nondominated solutions found ($Sol.$).

As expected, the results show that, for all instances, the number of nondominated solutions found was the same for all algorithms. However, when the consumed times are compared, the **mc-ParGRASP1** algorithm outperformed the others.

The **mc-ParGRASP2** algorithm reduces the idleness time of the **mc-ParGRASP1** algorithm. However, it consumes more time sending, to the master process, a total number of solutions greater than the necessary. A nondominated solution T at iteration y can be dominated by a solution T' found at iteration y+1. The sending of T was unnecessary.

Experiments were done varying the value of z (interval to send nondominated solutions) for the **mc-ParGRASP2** algorithm and it was verified that, for instances with r = 3 criteria, the consumed time decreases according to the value of z increases, obtaining an equal consumed time to the **mc-ParGRASP1** algorithm when $z = N\_iter$. In this

Table 1: Results of mc-ParGRASP1 and mc-ParGRASP2 algorithms on complete graphs with n = 20, 30 and 50 nodes and r = 2 and 3 criteria.

INSTANCE	n	r	MC-PARGRASP1		MC-PARGRASP2	
			SOL.	T(S)	SOL.	T(S)
g20_2	20	2	125	9	125	8
g20_3	20	3	4763	34	4763	41
g30_2	30	2	249	13	249	10
g30_3	30	3	12971	89	12971	170
g50_2	50	2	528	15	528	16
g50_3	50	3	28266	188	28266	401

situation, the **mc-ParGRASP2** algorithm is equivalent to the **mc-ParGRASP1** algorithm.

With the goal of decentralizing the procedure of updating the global Pareto list *lPareto*, the **mc-ParGRASP3** algorithm also sent a number of solutions greater than the necessary to the master process, generating a superior consumed time when compared with the other algorithms.

It was verified, in the **mc-ParGRASP1** algorithm, that the processes do not consume the same computational time to execute the *N_iter/p* iterations. Some of them finish before the others. Based on this observation, a variation of this algorithm is proposed. In this new algorithm, called **mc-ParGRASP1_wT** (**mc-ParGRASP1** *with termination*), each process i ($1 \leq i \leq p$) has a logic vector ψ_i, where each position j ($1 \leq j \leq p$) of this vector is **true** if the process j has already finished the *N_iter/p* iterations and **false**, otherwise. When a process i finishes the *N_iter/p* iterations, it sends a communication message to the other processes and verifies if the vector ψ_i is with **true** at all positions. If it is true, the process i starts to send the local Pareto list $lPareto_i$ to the master process (process 1). Otherwise, the process i executes other iteration and, at the end of it, it repeats the verification at vector ψ_i.

The previous experiment was executed again with the **mc-ParGRASP1** and **mc-ParGRASP1_wT** algorithms. Table 3 presents the results. For the instances with r = 2 criteria, where the number of nondominated solutions found is smaller, the consumed time was similar. For the instances with r = 3 criteria, the consumed time of the **mc-ParGRASP1_wT** algorithm was superior. It can be explained by the greater number of nondominated solutions obtained. It demands a greater computational time to send the solutions to the master process.

In another experiment, the **mc-ParGRASP1** and **mc-**

Table 2: Results of mc-ParGRASP1 and mc-ParGRASP3 algorithms on complete graphs with n = 20, 30 and 50 nodes and r = 2 and 3 criteria.

INSTANCE	n	r	MC-PARGRASP1		MC-PARGRASP3	
			SOL.	T(S)	SOL.	T(S)
g20_2	20	2	125	9	125	10
g20_3	20	3	4763	34	4763	91
g30_2	30	2	249	13	249	8
g30_3	30	3	12971	89	12971	293
g50_2	50	2	528	15	528	25
g50_3	50	3	28266	188	28266	456

Table 3: Results of mc-ParGRASP1 and mc-ParGRASP1_wT algorithms on complete graphs with n = 20, 30 and 50 nodes and r = 2 and 3 criteria.

INSTANCE	n	r	mc-ParGRASP1		mc-ParGRASP1_WT	
			SOL.	T(S)	SOL.	T(S)
g20_2	20	2	125	9	125	9
g20_3	20	3	4763	34	4911	40
g30_2	30	2	249	13	249	12
g30_3	30	3	12971	89	13194	104
g50_2	50	2	528	15	528	15
g50_3	50	3	28266	188	29790	205

ParGRASP1_wT algorithms were executed varying the number p of processors. It was used the larger instance, *g50_3*. Table 4 presents, for each algorithm, the number of nondominated solutions found (*Sol.*), the consumed time in seconds (t) and the speedup (*Sp.*). The speedup was calculated for k ($1 \leq k \leq 8$) processors using the ratio t_1/t_k, where t_1 is the consumed time by the algorithm using one processor and t_k is the consumed time using k processors.

The results show that according to the number p of processors increases, the difference between the consumed times increases. However, the difference between the numbers of nondominated solutions found increases too, achieving 2218 solutions (increase of 7.86%) with $p = 8$. The increase of the consumed time can be explained by the increase of the number of nondominated solutions.

CONCLUSION

The use of the GRASP metaheuristic for multi-criteria problems is recent (ARROYO *et al.*, 2005; VIANNA; ARROYO, 2004). We believe that the first parallel multi-criteria GRASP strategies are being proposed in this work.

The strategy described on **mc-ParGRASP1** algorithm presented the best results for the mc-MST problem. A variation of this algorithm, called **mc-ParGRASP1_wT**, obtained good computational times and improved the global Pareto list found by the sequential version (**mc-GRASP**).

The results show that parallelizing a multi-criteria GRASP algorithm reduces execution time and can also improve the set of nondominated solutions obtained by the sequential version.

Table 4: Results of mc-ParGRASP1 and mc-ParGRASP1_wT on a complete graph with 50 nodes and 3 criteria.

P	mc-ParGRASP1			mc-ParGRASP1_WT		
	SOL.	T(S)	SP.	SOL.	T(S)	SP.
1	28251	713	1.00	28251	715	1.00
2	28251	361	1.98	28449	378	1.89
4	28251	188	3.79	29790	205	3.49
8	28251	160	4.46	30433	180	3.97

Artigo recebido em 26/01/2006
Aprovado para publicação em 28/11/2006

References

ARROYO, J. E. C., VIEIRA, P. S. & VIANNA, D. S. (2005). A GRASP algorithm for the multi-criteria minimum spanning tree problem. In: *Second Multidisciplinary Conference on Scheduling: Theory and Applications*, Nova York, p. 1-11.

DRUMMOND, L. M. A., OCHI, L. S. & VIANNA, D. S. (2001). An asynchronous parallel metaheuristic for the period vehicle routing problem. *Future generation computer systems*, 17, p. 79-386.

EHRGOTT, M. & KLAMROTH, K. (1997). Connectedness of efficient solutions in multiple criteria combinatorial optimization. *European Journal Operations Research*, v. 97, p. 159-166.

EHRGOTT, M. & GANDIBLEUX, X. (2000). A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22, p. 25-460.

FEO, T. A. & RESENDE, M. G. C. (1995). Greedy randomized adaptive search procedures. *Global Optimization*, 6, p. 109-133.

GLOVER, F. & LAGUNA, M. (1997). *Tabu search*. Kluwer Academic Publishers.

HAMACHER, H. W. & RUHE, G. (1994). On spanning tree problems with multiple objectives. *Annals of Operations Research*, 52, p. 209-230.

HOLLAND, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

JONES, D. F., MIRRAZAVI, S. K. & TAMIZ, M. (2002). Multi-objective metaheuristics: An overview of the current state-of-art. *European Journal Operations Research*, 137, p. 1-19.

KIRKPATRICK, S., GELLAT JR., C.D. & VECCHI, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220, p. 671-680.

References

KRISHNAMOORTH, M., ERNST, A. T. & SHARAIHA, Y. M. Comparison of Algorithms for the Degree Constrained Minimum Spanning Tree. *Journal of Heuristics*, v. 7, p. 587-611, 2001.

KRUSKAL, J. B. On the shortest spanning tree of graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, v. 7, p. 48-50, 1956.

KNOWLES, J. D. Local search and hybrid evolutionary algorithms for Pareto optimization. Thesis of Doctorate, University of Reading, UK, 2002.

MOON, J. W. Various Proofs of Cayley's Formula for Counting Trees. In: *A Seminar on Graph Theory* [edited by F. Harary], New York: Holt, Rinehart and Winston, p. 70-78, 1967.

MURATA, T., ISHIBUCHI, H. & GEN, M. (2001). Specification of genetic Search directions in cellular multi-objective genetic algorithms. *Evolutionary Multi-Criterion optimization, EMO. LNCS*, 1993, p. 82-95.

OCHI, L. S., VIANNA, D. S., DRUMMOND, L. M. A. & VICTOR, A. O. A parallel evolutionary algorithm for the vehicle routing problem with heterogeneous fleet. *Future Generation Computer Systems*, 14, p. 285-292, 1999.

RAMOS, R. M., ALONSO, S., SICÍLIA, J. & GONZÁLES, C. (1998). The problem of the optimal biobjective spanning tree problem. *European Journal Operations Research*, v. 111, p. 617-628.

VIANNA, D. S., OCHI, L. S. & DRUMMOND L. M. A. A parallel hybrid evolutionary metaheuristic for the period vehicle routing problem. *Lecture notes in computer science*, 1586, p. 183-191, 1999.

VIANNA, D. S. & ARROYO, J. E. C. A GRASP algorithm for the multi-objective knapsack problem. In: *XIV International Conference of the Chilean Computer Science Society*, Arica, IEEE CS Press, p. 69-75, 2004.

Zhou, G. & Gen, M. Genetic algorithm approach on Multi-criteria minimum spanning tree problem. *European Journal Operations Research*, v. 114, p. 141-152, 1999.

Acknowledgments

This work was funded by the Municipal Town Hall of Campos dos Goytacazes city. The computational experiments were done at the Laboratório Nacional de Computação Científica – LNCC.

Sobre os autores

Dalessandro Soares Vianna
Professor Adjunto da Universidade Candido Mendes, UCAM-Campos,
Núcleo de Pesquisa e Desenvolvimento em Informática,
End.: Anita Peçanha, 100 – Parque São Caetano – Campos dos Goytacazes – RJ – 28040-320 – Brasil.
Tel. / Fax: (22) 2733-4100
E-mail: dalessandro@ucam-campos.br

José Elias Claudio Arroyo
Professor Adjunto da Universidade Candido Mendes, UCAM-Campos,
Núcleo de Pesquisa e Desenvolvimento em Informática,
End.: Anita Peçanha, 100 – Parque São Caetano – Campos dos Goytacazes – RJ – 28040-320 – Brasil.
Tel. / Fax: (22) 2733-4100
E-mail: jclaudio@ucam-campos.br

Pedro Sampaio Vieira
Aluno da Universidade Candido Mendes, UCAM-Campos,
Núcleo de Pesquisa e Desenvolvimento em Informática,
End.: Anita Peçanha, 100 – Parque São Caetano – Campos dos Goytacazes – RJ – 28040-320 – Brasil.
Tel. / Fax: (22) 2733-4100
E-mail: pedro.s.v@gmail.com

Thiago Ribeiro de Azeredo
Aluno da Universidade Candido Mendes, UCAM-Campos,
Núcleo de Pesquisa e Desenvolvimento em Informática,
End.: Anita Peçanha, 100 – Parque São Caetano – Campos dos Goytacazes – RJ – 28040-320 – Brasil.
Tel. / Fax: (22) 2733-4100
E-mail: thiagoribeiro@gmail.com