Sequential and Parallel Metaheuristics based on GRASP and VNS for Solving the Traveling Purchaser Problem

Luiz Satoru Ochi, Leonardo Soares Vianna,
Mozar B. da Silva and Lúcia M. de Assumpção Drummond

Department of Computer Science
Fluminense Federal University
R. Passo da Pátria, 156, Bloco E, Niterói, 24210-240, BRAZIL
e-mail: {lucia,satoru}@dcc.ic.uff.br

May 16, 2002

Abstract

This paper presents several strategies for sequential and parallel implementations of the Greedy Randomized Adaptive Search Procedure (GRASP) and the Variable Neighborhood Search (VNS) applied to a combinatorial optimization problem known as Traveling Purchaser Problem (TPP). The high potential of these metaheuristics and their hybrid versions (GRASP+VNS) is shown through the comparison with an algorithm Tabu Search for the TPP, that has presented the best results for this problem so far.

Keywords: Metaheuristics, Distributed Algorithms, Optimization

1 Introduction

Metaheuristics such as Genetic Algorithm (GA), Greedy Randomized Adaptive Search Procedures (GRASP),Tabu Search (TS) and Variable Neighborhood Search (VNS) have been used successfully for solving hard combinatorial optimization problems.

Although metaheuristics aim to eliminate or reduce historical difficulties of conventional construction and local search heuristics, such as premature stops in local optima solutions distant from an optimal solution in optimization problems, they may require a large amount of time to find good upper bounds as penalty in many cases. This fact has motivated the developing of parallel metaheuristics, taking advantage of the inherent parallelism present in some of them such as GRASP and GA [1] [4] [11].

Many parallel algorithms of Tabu Search have also been developed [2] [3] probably because of the excellent results obtained by sequential versions of TS algorithms.

We intend to show in this paper that metaheuristics such as GRASP and VNS can achieve good results comparable to the best versions of TS, concerning not only the quality of solutions but the required time as well for solving a generalization of the Traveling Salesman Problem (TSP) called Traveling Purchaser Problem (TPP). This paper proposes new sequential and parallel algorithms based on GRASP and VNS concepts. These algorithms were run on an IBM SP/2 computer using MPI in the parallel algorithms [15].

The remainder of this paper is organized as follows. Section 2 describes the TPP. Section 3 presents the sequential and parallel algorithms proposed based on GRASP and VNS. Computational results are shown in Section 4. Finally, Section 5 concludes the paper.

2 The Traveling Purchaser Problem

The TPP is classified as NP-Hard and can be seen as a generalization of the Traveling Salesman Problem (TSP). To describe the TPP we need the following data: a set of n markets $M = \{1, 2, \cdots, n\}$ plus a source $s = \{0\}$; a set of m items $K = \{1, 2, \cdots, m\}$ to be purchased at n markets; an array $P = (p_{kj})$ such that $k \in K, j \in M$, where p_{kj} is the cost of item k at market j and an array $T = (t_{ij})$ such that $i, j \in M$, where t_{ij} is the cost of travel from i to j.

It is assumed that each item is available in at least one market $j \in M$; in source $s = \{0\}$ no item can be purchased; the traveler may pass through a market any number of times without purchasing an item there; and the traveler may purchase as many items as there are available at each market. There is a complete symmetrical graph $G = (M, E)$, without loops, where M is the already defined set of markets and each edge $(i, j) \in E$ represents a link between i and j, such that $i, j \in M$. The aim of the TPP is to obtain a directed cycle including source s and passing through a subset $J \subseteq M$ such that the total travel and purchase costs are minimized.

Although TPP has been used in many applications such as scheduling and routing problems, it has not been extensively studied in related literature. TPP was originally developed by Ramesh [13] who proposed a method based on lexicographic search procedure. Golden, Levy and Dahl [7] proposed a heuristic based on saving and insertion concepts. Ong [10] presented a new heuristic based on the algorithm proposed by Golden, Levy and Dahl called " Tour - Reduction algorithm". Voss [16] [17] presented metaheuristics based on Dynamic Tabu Search for the TPP which used dynamic strategies for managing of tabu lists. Pearn and Chien [12] proposed algorithms based on "Commodity-Adding".

3 Algorithms based on GRASP and VNS

Metaheuristics based on concepts of GRASP [5] [1] [4] and VNS [9] [8] require two algorithms, one of them for generation of an initial solution and the other for local

search phase.

Initially, the goal of this work was the analysis of performance of algorithms GRASP and VNS using the same methods of construction and local search that were employed in the algorithms proposed by Voss [17]. Thus, the methods Add and Drop were implemented for construction and Drop for local search. Moreover, other algorithms of construction and of local search were also proposed.

The following algorithms were developed for construction of initial solutions:

- Add - it consists of a loop where at each step, the market which gives the best saving is inserted in the partial solution. The loop finishes when all items are available in the markets of the current solution.

- Drop - this algorithm begins from a feasible solution of the Traveling Salesman Problem (TSP), which includes all nodes (markets), and then it executes a loop. At each iteration the node which offers the best saving, in case of being taken away of the current solution, is removed, since all products keep available in the remaining markets. The initial route is defined using the algorithm GENIUS proposed for the TSP [6]. The loop finishes when it is not possible to obtain any saving by removing markets without violating the constraints of the problem.

- AddGeni - this algorithm is similar to Add, but the criterion of selection of the next node to be inserted is from the heuristic GENIUS [6].

- DropGeni - this algorithm is similar to Drop, but the criterion of selection of the node to be removed from the current solution is from the heuristic GENIUS [6].

- AddRandom - at each step of this algorithm a node is chosen at random to be inserted in the solution from a constrained candidate list (CCL), that is composed by k nodes (markets) which offers the best savings, in case of being inserted, where k is an entry parameter.

- DropRandom - it initiates with a solution containing all markets and at each step of a loop, a market is selected randomly from a constrained candidate list CCL, composed by k nodes which offer the best savings, in case of being removed, where k is an entry parameter.

The following algorithms for local search were also employed:

- AddSearch - it applies Add procedure on an initial solution while the solution is being improved [17].

- DropSearch - it applies Drop procedure on an initial solution while the solution is being improved [17].

- AddDropSearch - each node (market) that does not belong to the current solution, is inserted and the DropSearch procedure is executed.

- DropAddSearch - each node that belongs to the current solution, is removed and AddSearch procedure is used.

- SwapSearch - it swaps the positions of pair of nodes on the initial solution for y iterations, where y is an entry parameter.

- NeighSearch - it builds a list containing p elements chosen at random from the current solution. Then, these elements are removed from the solution and they are considered forbidden (they can not be part of the solution for some time). It executes one of the local search algorithms previously described. The p nodes (markets) are enabled again and it once more executes one of the local search algorithms previously described. All these steps are executed for y iterations, where y is an entry parameter.

- VNSSearch - this algorithm executes NeighSearch in a loop. Initially $k = 1$ and if the solution is not improved k is increased ($k \leftarrow k + 1$), otherwise it goes on using a neighborhood with $k = 1$.

- HybridSearch - this algorithm is a hybrid procedure including several local search algorithms: AddSearch; DropSearch; DropAddSearch; AddDropSearch and SwapSearch. This sequence of algorithms is repeated until no improved solution is reached.

3.1 Partial Computational Results

Initially all combinations of the procedures of construction and local search described in the previous section were included in several versions of GRASP and VNS algorithms.

Concerning the GRASP, each of the six algorithms for construction were adapted (when necessary), using a candidate chosen randomly from a constrained candidate list instead of the best candidate.

Concerning the VNS, the six algorithms for construction were combined with each algorithm for local search adapted in a structure VNS like with multiple neighborhoods N_i ($i = 1, \ldots, k$, where k represents the number of neighborhoods considered). Thus, for example, in the local search Drop, in the first neighborhood N_1, at each step only a node is removed of the current solution, in neighborhood N_2, two nodes are removed simultaneously at each step, etc. In the same way, we adapted the other algorithms for local search employing a structure with several neighborhoods.

To our best knowledge, there are not test problems for the TPP available in public sites. Therefore, in order to evaluate the algorithms proposed a set of instances for the TPP was generated randomly.

At first, all combinations of construction and local search procedures were implemented generating 48 versions of algorithms based on GRASP and 48 based on VNS. In order to analyze the performance of these algorithms, each of them was tested for 36 instances of TPP, where the number of markets varied from 50 until 150, the number of items from 50 until 150, the number of items per market from 1 to 5, the

cost of distances from 10 to 300 and finally the cost of items from 10 to 300. The stop criterion in all cases was 600 seconds. The programs were executed in a node of an IBM SP/2 computer dedicated exclusively to our tests. The time 600 seconds was chosen after several tests with different versions of GRASP and VNS. Each one of the 36 instances was executed 5 times.

We analyzed the performance of these 96 algorithms in the following way. For each instance i ($i = 1, \ldots, 36$), the best solution called $bs(i)$ obtained among the 96 algorithms is determined. The average error of each algorithm in each instance is determined by the difference in percent of the average of solutions of this algorithm and $bs(i)$. The average error for each algorithm represents the arithmetic average of the average errors considering the 36 instances. The best two algorithms GRASP called GRASP1 and GRASP2, employed for construction and local search the methods AddRandom - Hybrid and AddGeni - Hybrid, respectively. The best two algorithms VNS, called VNS1 and VNS2, used for construction and local search DropGeni - Hybrid and Drop - Hybrid, respectively.

In a second stage of this work, hybrid sequential algorithms were proposed combining GRASP with local search algorithms based on VNS. The two algorithms GRASP+VNS1 and GRASP+VNS2 that obtained the best average results in the initial tests, employed AddGeni and AddRandom for construction, and Hybrid for local search in VNS.

In order to evaluate the performance of our algorithms for solving TPP, we implemented also the algorithms Tabu Search proposed by Voss [17], because they have given the best results for this problem so far, according to the author. We implemented the two best versions of Tabu Search proposed by Voss, including intensification, diversification phases and REM (Reverse Elimination Method) and CSM (Cancellation Sequence Method) dynamic list management. These two versions are called TABU-CSM and TABU-REM and they use the algorithms Add and Drop for construction of initial solutions, the algorithm Drop for local search and CSM and REM for list management respectively. An average of these results are presented in Table 1 that shows that VNS and GRASP can obtain good results comparable to the results obtained by Tabu Search. In this table, "Time(s)" indicates the average time in seconds to obtain the best solution, "Total of best solutions" represents the number of times in percent that the algorithm achieved $bs(i)$ considering the 36 instances and "Average error (%)" is calculated as previously described.

Algorithms	Average Error (%)	Time (s)	Total de Best Solutions (%)
GRASP1	0.09	340.72	52.78
GRASP2	0.16	343.75	47.22
VNS1	0.19	318.56	47.22
VNS2	0.11	324.97	44.44
GRASP+VNS1	**0.04**	327.31	**63.89**
GRASP+VNS2	0.09	279.50	55.56
TABU+CSM	0.27	302.72	41.67
TABU+REM	0.41	**234.69**	50.00

Table 1: Average results of TS and the best two versions of GRASP, VNS and GRASP+VNS

The good performance of the sequential algorithms GRASP, VNS and GRASP+VNS

motivated the development of different strategies and models of parallel algorithms based on the sequential algorithms here proposed.

3.2 Strategies for Distributed Parallel Implementations of GRASP and VNS

In GRASP algorithms, iterations are independent from each other. Therefore, iterations may be easily shared among processors. The sharing of iterations among processors can be considered a load balance problem. In order to balance the iterations among processors we used two strategies: static load balance and dynamic load balance.

In static load balance model, the number of iterations that each process executes is determined before the execution. In the proposed algorithms, the number of iterations of the associated sequential algorithm was divided equally by the processes that take part of the parallel algorithm. Thus, each process executes $\lceil itseq/proc \rceil$, where $itseq$ represents the number of iterations executed by the sequential algorithm and $proc$ represents the number of processes in the distributed algorithm. Each process executes independently from each other.

In dynamic load balance model, the number of iterations each process executes is determined during the execution. This kind of balance aims to prevent a slow process from increasing the execution time of the distributed program. It is obtained by giving more iterations to faster processes. Two asynchronous algorithms for dynamic load balance were implemented. One of them uses the master-worker model while the other is completely distributed.

In the master-worker model, in the beginning of the execution all worker processes receive a small fixed number of iterations. When a process completes its initial iterations, it sends a message to the master informing this fact. The master either sends more iterations to the worker or a message of termination.

We propose in this paper a completely distributed parallel GRASP based on dynamic load balance. In the completely distributed model, for each process that executes the GRASP algorithm, called p_i, there is another process associated (sharing the same processor), called q_i (for $1 \leq i \leq w$, where w is number of processors), that is responsible for controlling the number of iterations of the corresponding process p_i and for termination of the distributed program. Process q_i stays idle most of time, therefore it does not compete with p_i for the CPU very much.

Initially the iterations are divided equally among the w processors. Each process p_i executes the algorithm for an initial number of iterations. When it finishes, it asks for more iterations to q_i. If q_i has available iterations it gives them to p_i. Otherwise, q_i asks for more iterations to another process q_j (such that $1 \leq j \leq w$ and $j \neq i$), chosen randomly. Upon receiving extra iterations, process q_i forwards them to the corresponding process p_i. If q_i does not receive extra iterations, it goes on questioning other processes until it receives the iterations or it receives refusals from all other processes. In the latter, q_i sends a message of termination to the corresponding process p_i and finishes its execution. Process p_i finishes its execution upon receiving

a message of termination from the associated q_i.

The distributed parallel models applied to VNS are akin to those used in GRASP. So, each process executes a VNS algorithm and the iterations that compose the main loop of this algorithm is divided by the processes. In the first strategy, static load balance is employed and each process is executed independently. The second strategy uses the master-worker model where there is a master process that is responsible for the distribution of the iterations among the worker processes. In the third strategy, a completely distributed model is applied, where for each process p_i that executes the VNS algorithm, there is a corresponding process q_i, that manages the iterations.

The distributed parallel strategies implemented were called ParInd for the distributed algorithm that executes static load balance, ParDist for the algorithm that uses dynamic load balance in a completely distributed model and ParMW for the algorithm that uses dynamic load balance in a master-worker model.

4 Computational Results

The parallel strategies described previously were implemented on five algorithms. In the first one, called GDif, each processor executes a different version of GRASP. In the second one, GEqual, every processor executes the same algorithm GRASP, that presented the best results in the sequential version employing AddRandom for construction of an initial solution and Hybrid for local search. In the same way, in VDif each processor executes a different version of VNS and in VEqual, each processor executes the same VNS employing Drop for generation of an initial solution and Hybrid for local search. Finally in GVDif, each processor executes a different algorithm based on GRASP+VNS.

These algorithms were executed on an IBM SP/2 using MPI for parallelism with 4 and 8 processors dedicated exclusively to the application.

We generated five instances randomly varying the number of markets from 100 to 500, the number of items from 100 to 500, the cost of travel from 10 to 500, the number of items per market from 5 to 100 and the cost of each item from 10 to 500. Each program was executed 3 times for each instance and in all cases 2000 iterations were executed.

In tables 2 and 3, the average results of GVDif, that presented the best average results when compared with the other algorithms (GDif, GEqual, VDif, VEqual) considering all proposed strategies for parallel implementations, are shown. In these tables, the average cost ("Cost"), average time in seconds ("Time"), speedup ("SU") and efficiency ("E") for four ("4P") and eight ("8P") processors are presented. The cost and time of the sequential algorithms are obtained using only one processor. Remark that speedup measures the acceleration observed for the parallel algorithm when compared with its sequential version and efficiency measures the average fraction of the time along which each process is effectively used. Thus, SU(P)=T(seq)/T(P), such that T(seq) is the time required for the sequential algorithm and T(P) the time required for the parallel algorithm run on P processors, and E(P)= SU(P)/P.

In GVDif, each process executes a hybrid sequential algorithm that consists of

a GRASP procedure that applies a local search method based on VNS. Thus, the algorithms employed for construction of initial solutions of GVDif are AddGeni and AddRandom, when 4 processors are used. In case of 8 processors, DropGeni and DropRandom were also executed. The local search phases employ Hybrid and DropAdd algorithms adapted in a structure with several neighborhoods as previously described. The parallel algorithms are compared with the sequential version that consists of a GRASP+VNS algorithm whose procedures of construction and local search vary according to the iteration, i.e., each strategy (AddGeni - VNS Hybrid, AddRandom - VNS Hybrid, AddRandom - VNS DropAdd, AddGeni - VNS DropAdd,DropGeni - VNS Hybrid, DropRandom - VNS Hybrid) is executed for an equal number of iterations. The algorithm GVDif reduced the execution times significantly improving also the quality of solutions in most cases when compared with the previous algorithms.

Algorithm	Cost(4P)	Time(4P)	Cost(8P)	Time(8P)
Sequential	**17525.106**	18790.667	**17525.106**	18790.667
ParInd	17612.509	**5984.350**	17561.133	**2355.900**
ParDist	17586.838	6247.898	17549.562	2393.208
ParMW	17609.722	7933.148	17563.103	2804.362

Table 2: Average Cost and Time of Algorithm GVDif

Algorithm	SU(4P)	E(4P)	SU(8P)	E(8P)
ParInd	**3.14**	**0.78**	**7.98**	**0.99**
ParDist	3.01	0.75	7.85	0.98
ParMW	2.37	0.59	6.70	0.84

Table 3: Speedup (SU(.)) and Efficiency (E(.))of Algorithm GVDif

In all programs executed, the increasing of the number of processors improves the qualities of solutions, because more processors explore the search space. The increasing of the number of processors improves also the efficiency in all cases. In ParInd this happens because the number of iterations received by each process is inversely proportional to the number of processors employed. As each process may execute different procedures of local search and construction that may present different execution times, the delay impact is more evident when the number of iterations is increased. Thus when more processors are used, the delay caused by a process running a slower algorithm is smaller than when it executes more iterations, that happens when less processors are employed.

In ParDist and ParMW the increasing of efficiency can be explained by the fact that when the number of processors employed increases, the necessity of load balance reduces and less messages are exchanged, increasing the efficiency.

Although the algorithms proposed presented good results, their versions that use dynamic load balance did not present the best results considering execution times. This happens because the tests were executed on a computer dedicated exclusively to the application with identical processors and the different times required by the several algorithms of construction and local search did not compensate for the time wasted with load balance in this computational system.

5 Concluding Remarks

In this paper, several algorithms based on concepts of GRASP and VNS for the TPP were proposed. One of the main goals of this work was to show that algorithms GRASP and VNS can give results as good as the best versions of Tabu Search algorithms existing in literature. We employed several algorithms of construction and local search in our algorithms GRASP and VNS, including some that were employed for the Tabu Search algorithm proposed by Voss [16] [17].

The high potential of the algorithms GRASP, VNS and GRASP+VNS could be testified through the good computational results obtained by the sequential algorithms proposed. This good performance encouraged the development of several strategies of parallelism for these metaheuristics, aiming the analysis of the parallel models more appropriate for GRASP and VNS algorithms.

Thus, we proposed several strategies for parallel implementations of the Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Search (VNS) applied to the TPP, based on independent, master-worker and completely distributed models, using static and dynamic load balance. Although the algorithms proposed presented good results, their versions that use dynamic load balance did not present the best results because the tests were executed on a computer dedicated exclusively to our application with identical processors. We could observe that when the number of processors increased the necessity of load balance reduced and consequently less messages were exchanged and the efficiency increased. The increasing of the number of processors caused the improvement of the qualities of solutions in all programs too. Finally we observed that the algorithm that employed concepts of GRASP and VNS, GVDif, presented the best results concerning the execution times and the efficiency in the parallel models, improving also the quality of solutions in most cases. This fact indicates that the use of this hybrid approach may be very advantageous.

Experimental tests will continue on a heterogeneous system in order to further evaluate the dynamic load balance strategy.

References

[1] AIEX, R.M., BINATO, S. and RESENDE, M.G.C, 2002, Parallel GRASP with Path-relinking for Job Shop Scheduling. To appear in Parallel Computing.

[2] CRAINIC, T. G.; TOULOUSE and M. and GENDREAU, M., Toward a Taxonomy of Parallel Tabu Search Heuristics, 1996, *INFORMS Journal on Computing*.

[3] CRAINIC, T. G.; TOULOUSE , M. and GENDREAU, M., 1996, Parallel Asynchronous Tabu Search for Multicommodity Location-allocation with Balancing Requirements, *Annals of Operations Research 63*, 277–299.

[4] CUNG, V. D.; MARTINS, S.L.; RIBEIRO, C. C. and ROUCAIROL, C., 2001, Strategies for the Parallel Implementation of Metaheuristics, Essays and Surveys in Metaheuristcs, *Kluwer Academic Publishers*, 263-308.

[5] FEO, T. A. and RESENDE, M. G. C., 1995, Greedy Randomized Adaptive Search Procedures, *Journal of Global Optimization*, Vol 6, 109-133.

[6] GENDREAU, M.; HERTZ, A. and LAPORTE, G., 1992, New Insertion Post-optimization Procedures for the Traveling Salesman Problem, *Operations Research*, Vol 40,1086-1094.

[7] GOLDEN, B.; LEVY, L. and DAHL, R.,1981, Two Generalizations of the Traveling Salesman Problem, *OMEGA 9*, 439-455.

[8] HANSEN, P.; MLADENOVIĆ, N. and PEREZ-BRITO, D., 1998, Variable Neighborhood Decomposition Search, *Les Cahiers du GERAD G-98-53*, University of Montreal.

[9] MLADENOVIĆ, N., 1995, A Variable Neighborhood Algorithm - a New Metaheuristic for Combinatorial Optimization, *Abstract of papers presented at Optimization Days*, Montreal, 12.

[10] ONG, H. L., 1982, Approximate Algorithms for the Traveling Purchaser Problem, *Operations Research Letters 1*, 201-205.

[11] DRUMMOND, L. M. A.; OCHI, L. S. and VIANNA, D. S. 2001, An Asynchronous Parallel Metaheuristic for the Period Vehicle Routing Problem, *Future Generation Computer Systems 17*, 379-386.

[12] PEARN, W. L. and CHIEN, R. C., 1998, Improved Solutions for the Traveling Purchaser Problem, *Computer and Operations Research*, Vol 25(11), 879-885.

[13] RAMESH, T., 1981, Traveling Purchaser Problem, *Opsearch*, 18, 78-91.

[14] RESENDE, M. G. and FESTA, P, 2001, Grasp: An Annotated Bibliography, Essays and Surveys in Metaheuristcs, *Kluwer Academic Publishers*, 325-368.

[15] SNIR, M.; OTTO, S. W.; HUSS-LEDERMAN, S.; WALKER, D. W. and DONGARRA, J., 1996, MPI: The Complete Reference, *The MIT Press*.

[16] VOSS, S., 1996, ADD and DROP-procedures for the Traveling Purchaser Problem, *Methods of operations research*, 53, 317-318.

[17] VOSS, S., 1996, Dynamic Tabu Search Strategies for the Traveling Purchaser Problem, *Annals of Operations Research*, 63, 253-275.