Towards Grid Implementations of Metaheuristics for Hard Combinatorial Optimization Problems

Aletéia P.F. Araújo, Sebastián Urrutia
Department of Computer Science, Catholic University of Rio de Janeiro
Marquês de São Vicente 225, Rio de Janeiro, RJ 2453-900, Brazil
{aleteia,useba}@inf.puc-rio.br

Cristina Boeres, Vinod E.F. Rebello, Celso C. Ribeiro
Instituto de Computação, Universidade Federal Fluminense
Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil
{boeres, vinod, celso}@ic.uff.br

Abstract

Metaheuristics are approximation algorithms that find very good solutions to hard combinatorial optimization problems at the expense of large computational requirements. They do, however, offer a wide range of possibilities for implementations of effective robust parallel algorithms which run in much smaller computation times. We present four strategies for the parallelization of an extended GRASP with ILS heuristic for the mirrored traveling tournament problem. Computational results on widely used benchmark instances using various processors illustrate the effectiveness and the scalability of the different parallel strategies. These low communication cost parallel heuristics produce better quality solutions than the best known sequential algorithm.

1. Motivation

The organization and management of sporting events and championships is a worldwide multibillon dollar industry. Schedules with minimum traveling times and offering similar conditions to all teams taking part in a competition are of major interest for teams, leagues, sponsors, fans, and the media. In the case of the Brazilian national soccer championship, a single trip from Porto Alegre to Belém takes almost a full day's journey, with many stops due to the absence of direct flights, to cover a distance of approximately 4000 kilometers. The total distance traveled becomes an important variable to be minimized, so as to reduce traveling costs and to give the players more time to train and time off along the season that lasts for approximately eight months.

Several authors in different contexts (see e.g. [1, 12, 15, 22, 25, 26]) have tackled the problem of tournament scheduling in different leagues and sports such as soccer, basketball, hockey, baseball, rugby and cricket, using different techniques such as integer programming, tabu search, genetic algorithms, simulated annealing, and constraint programming.

The Traveling Tournament Problem is an inter-mural championship timetabling problem that abstracts certain characteristics of scheduling problems in sports [4]. It combines tight feasibility constraints with a difficult optimization problem. The objective is to minimize the total distance traveled by the teams, subject to the constraint that no team can play more than three consecutive games at home or away. Since the total distance traveled is a major issue for every team taking part in the tournament, solving a traveling tournament problem may be a starting point for the solution of real timetabling applications in sports, in general.

Metaheuristics are general high-level procedures that coordinate simple heuristics and rules to find good approximate (often optimal) solutions to computationally difficult combinatorial optimization problems. Among them, we find simulated annealing, tabu search, Greedy Randomized Adaptive Search Procedure (GRASP), genetic algorithms, scatter search, Variable Neighborhood Search, ant colonies, and others. They are based on distinct paradigms and offer different mechanisms to escape from locally optimal solutions, contrary to greedy algorithms or local search methods. Metaheuristics are among the most effective strategies for solving combinatorial optimization problems in practice and have been applied to a very large variety of areas and situations. The customization (or instantiation) of a metaheuristic to a given problem yields a heuristic for that prob-

lem.

Heuristics derived from metaheuristics are often time consuming methods that find very good solutions to hard optimization problems. Metaheuristics offer a wide range of possibilities for effective parallel algorithms running in much smaller computation times, but require efficient implementations. Cung *et al.* [3] noted that parallel implementations of metaheuristics not only appear quite naturally as an alternative to speed up the search for good approximate solutions, as also facilitate solving larger problems and finding improved solutions, with respect to their sequential counterparts, due to the partitioning of the search space and to the increased possibilities for search intensification and diversification. As a consequence, parallelism improves the effectiveness and robustness of metaheuristic-based algorithms. The latter are less-dependent on parameter tuning and their success is not limited to few or small classes of problems.

The growing computational power requirements of large scale applications and the high costs of developing and maintaining supercomputers has fueled the drive for cheaper high performance computing environments. With the considerable increase in commodity computers and network performance, cluster computing and, more recently, grid computing has emerged as a real alternative to traditional supercomputing environments for developing parallel applications that harness massive computational resources.

A computational grid [10] is the cooperation of distributed computer systems where user jobs can be executed both on local and remote computer systems, creating a virtual environment for solving large-scale applications, such as problems in combinatorial optimization. The complexity, however, incurred in writing such parallel grid-aware applications is higher than in traditional parallel computing environments. Therefore, developing and tuning efficient parallel implementations of metaheuristics in grid platforms requires a thorough programming effort.

This paper describes four efficient and simple strategies for the parallelization in grid environments of the extended GRASP with ILS (Iterated Local Search) heuristic for the mirrored traveling tournament problem proposed in [21]. The sequential strategy substitutes the local search phase of a GRASP heuristic by an ILS procedure, obtaining high-quality solutions that are among the best known in the literature for this problem [23].

The remainder of the paper is organized as follows. The following section reviews the formulation of the mirrored traveling tournament problem. Section 3 summarizes the extended GRASP with ILS sequential heuristic. In Section 4, some important issues concerning the parallel implementation of metaheuristics are introduced. Section 5 describes the four parallel implementations for the mirrored traveling tournament problem. Section 6 presents some preliminary experimental results obtained with the proposed strategies. Concluding remarks are presented in the last section.

2. The mirrored traveling tournament problem

Consider a tournament played by n teams, where n is an even number. In a *simple round-robin* (SRR) tournament, each team plays every other exactly once in $n-1$ prescheduled rounds. In a *double round-robin* (DRR) tournament, each team plays every other twice, once at home and once away. A *mirrored double round-robin* (MDRR) tournament is a simple round-robin tournament in the first $n-1$ rounds, followed by the same tournament with reversed venues in the last $n-1$ rounds. Assume that each team in the tournament has a stadium in its home city. The distances between the home cities are known. Each team is located at its home city at the beginning of the tournament, to where it returns at the end after playing the last away game. Whenever a team plays two consecutive away games, it goes directly from the city of the first opponent to the other, without returning to its own home city.

The Traveling Tournament Problem (TTP) was first established by Easton *et al.* [4]. Given n (even) teams and the distances between their home cities, the TTP consists in finding a DRR tournament such that every team does not play more than three consecutive home or away games, no repeaters (i.e., two consecutive games between the same two teams at different venues) occur, and the sum of the distances traveled by the teams is minimized. Benchmark instances are available in [23]. To date, even small benchmark instances of the TTP with $n = 10$ teams cannot be exactly solved. The largest instance solved exactly to date for $n = 8$ teams took four days of processing time using twenty processors in parallel [5]. We refer to this problem as the non-mirrored TTP, for which both mirrored and non-mirrored solutions are feasible.

The mirrored Traveling Tournament Problem (mTTP) has an additional constraint: the games played in round k are exactly the same played in round $k + (n-1)$ for $k = 1, \ldots, n-1$, with reversed venues.

3. Extended GRASP with ILS heuristic

The GRASP (Greedy Randomized Adaptive Search Procedure) metaheuristic [16] is a multi-start or iterative process, in which each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is investigated during the local search phase until a local minimum is found. The best overall solution is kept as the result.

The construction and local search phases are problem-dependent and should be customized for each problem. GRASP has experimented continued development and has been applied in a wide range of areas [8]. Resende and Ribeiro [16, 17] described successful implementation techniques and parameter tuning strategies, as well as enhancements, extensions, and hybridization of the original algorithms.

The ILS (Iterated Local Search) metaheuristic [13] starts from a locally optimal feasible solution. A random perturbation is applied to the current solution and followed by local search. If the local optimum obtained after these steps satisfies some acceptance criterion, then it is accepted as the new current solution, otherwise the latter does not change. The best solution is, if necessary, updated and the above steps are repeated until some stopping criterion is met.

A hybridization of the GRASP and ILS metaheuristics into an effective hybrid heuristic for the mTTP was proposed in [21]. Basically, the authors substitute the local search phase of GRASP by an ILS procedure. The pseudo-code in Algorithm 1 summarizes the main steps of the `GRILS-mTTP` heuristic for finding approximate solutions for the mirrored traveling tournament problem.

Procedure `GRILS-mTTP`();
Result : Solution S^*
while $StoppingCriterion$ **do**
  $S \leftarrow$ `BuildGreedyRandomizedSolution`();
  $\underline{S}, S \leftarrow$ `LocalSearch`(S);
  repeat
    $S' \leftarrow$ `Perturbation`(S);
    $S' \leftarrow$ `LocalSearch`(S');
    $S \leftarrow$ `AcceptanceCriterion`(S, S');
    $S^* \leftarrow$ `UpdateGlobalBestSolution`(S, S^*);
    $\underline{S} \leftarrow$ `UpdateIterationBestSolution`($S, \underline{S}$);
  until $ReinitializationCriterion$;
end

Algorithm 1: Pseudo-code of the GRASP with ILS heuristic for the mTTP.

The outer **while** loop in Algorithm 1 executes a GRASP *construction phase* followed by an ILS *local search phase*, until a stop criterion is met. During the GRASP phase of each iteration, an initial solution S is constructed to which a local search algorithm is then applied, returning a new current solution S. This solution is also used to initialize the best solution $\underline{S}$ in the current iteration.

The ILS phase of the iteration is the inner **repeat** loop which applies a perturbation to the current solution S obtaining a new solution S'. A local search algorithm is applied to S', where four neighborhood structures are used. The first three are simple exchanges in which TS (team swap), HAS (home-away swap) and PRS (partial round swap) neighborhoods are explored by local searches. The GR (game rotation) ejection chain neighborhood, explored only as a diversification move, is performed less frequently by the heuristic as a perturbation.

A first-improving strategy similar to the VND (Variable Neighborhood Descent) procedure [11] was used to implement the local search algorithm. Once a local optimum with respect to the TS neighborhood is found, a quick local search using the HAS neighborhood is performed. Next, the PRS neighborhood is investigated, followed again by a local search using the HAS neighborhood. This scheme is repeated until a local optimum with respect to these three neighborhoods is found.

In this context, the new solution S' is accepted or not as the new current solution, depending on the result of an acceptance criterion. The best overall solution S^* and the best solution in the current GRASP iteration are updated, if necessary, and a new cycle starts with the perturbation of the current solution, until a reinitialization criterion is met.

A new GRASP iteration starts if 50 deteriorating moves have been accepted since the last time $\underline{S}$ (the best solution found in this GRASP iteration) was updated. Reinitialization occurs if too many perturbations followed by local search are performed without improving the best solution in the current GRASP iteration. It is important to notice that a GRASP iteration is not interrupted if the current solution S is still being improved.

The parallelization of this algorithm does not only aim to reduce the total running time, but also to improve its effectiveness and robustness. The use of several processors concurrently to explore different search trajectories, as described later, may lead to a more thorough investigation of the neighborhoods.

4. Parallel implementation of metaheuristics

One of the programming paradigms commonly used to develop parallel programs on distributed clusters is the *master-slave* approach (also often referred to as task farming) [9]. This approach is specially attractive, since it can generally be applied to take advantage of all available resources in a grid environment. Cung *et al.* [3] reviewed some major issues on parallel implementations of metaheuristics, such as the types of parallelism as well as appropriate parallel programming models and parallelization strategies.

Concerning parallelization strategies [3, 24], two main approaches are used: single-walk and multiple-walk. Each iteration of a metaheuristic generally starts with the construction of an initial solution, followed by a search to improve on the solution. The new neighboring solutions are evaluated by making a series of minor alterations to a given solution. The sequence of solutions evaluated is known as a

walk or *trajectory*. In the case of a single-walk parallelization, one unique search trajectory is traversed in the solution space and the search for the best neighbor at each iteration is performed in parallel. The neighborhood search is performed faster in parallel, but the search trajectory is the same as the one followed in the corresponding sequential implementation. On the other hand, a multiple-walk parallelization strategy is characterized by the investigation in parallel of multiple trajectories, each of them performed by a different processor. A search "thread" is a process running in each processor traversing a walk in the solution space. These processes can be either independent (where no information is exchanged among processes) or cooperative (the information collected along a trajectory is disseminated and used by other processes to improve or to speed up the search).

Cooperative strategies are the most general and promising, but often incur in additional costs in terms of communication and storage. However, if cooperation is well explored and implemented, it can globally lead to better solutions in smaller computation times even if each individual iteration may take longer, see e.g. [19]. Developing and tuning efficient parallel implementations of metaheuristics require a thorough programming effort and more implementation skills. The most difficult aspects to be determined are the nature of the information to be shared, in order to improve the search without taking too much additional memory or time to be collected, as well as the frequency at which this information is exchanged.

5. Parallel strategies for the extended GRASP with ILS heuristic

This section presents four simple, but efficient strategies, for the parallelization of the best known algorithm (the hybrid metaheuristic GRILS-mTTP [21] summarized in Section 3) for solving the mTTP. Besides obtaining speedups in execution times, an improvement in the quality of the solutions is also sought. All four versions are based on the Master-Worker programming paradigm and adopt a multiple-walk search strategy. This work aims to investigate how degrees of cooperation and increased diversity (in terms of number of trajectories investigated and the amount of information being shared) affect the GRILS-mTTP metaheuristic.

Initially, the master process generates and distributes distinct seeds to be used by the pseudo-random number generator of each worker process. As the number of workers increases, this will foster greater diversity. In order to reduce the chance that processes search the same neighborhood (i.e. evaluate the same solutions), each process uses a different sequence of pseudo-random numbers. The Mersenne Twister random number generator of Matsumoto and Nishimura [14] was chosen based on the recommendation in [20].

5.1. Parallel strategy with independent processes

This version, denoted by *PAR-I*, is representative of executing the sequential algorithm simultaneously on multiple machines (e.g. parameter sweep application). After receiving their seeds, each worker starts a cycle in which it generates a new solution during a GRASP construction phase and then executes an ILS local search phase until the reinitialization criterion is met. This cycle is repeated until a solution with a cost equal or better than the given target is found, in which case this solution is sent immediately to the master. The master, on receiving a target solution, broadcasts a halt message to each worker for them to finish their execution. Given the fact that no communication occurs between the master and workers, the master also participates as a worker process.

5.2. Parallel strategy with one-off cooperation

This version, *PAR-O*, is identical to *PAR-I* with the exception of the first iteration of the main loop. After each worker executes the GRASP phase, the best initial solution encountered by each is sent to the master, which in turn selects and broadcasts back to all the workers the best overall solution. Therefore, all workers will execute the ILS local search phase of the first iteration using the same initial solution. The following iterations are executed independently. Again, in this version, the master also continues to participate as a worker process.

This is called one-off cooperation because this exchange only occurs during the first iteration. The basis for this version comes from the fact that research has shown that searches which begin with good solutions will converge faster [6] particularly when searching multiple trajectories in parallel.

5.3. Parallel strategy with one elite solution

One of the possible shortcomings of the previous versions is the lack of continuous cooperation between workers during their execution, i.e. each worker processes does not learn from searches carried out in parallel (or solutions found) in previous iterations by other workers.

In the earlier strategies, the current best solution is not available to all workers. Information gathered from good solutions should be used to implement more effective strategies [7, 18]. Typically, in these history-based parallel cooperative strategies, the master manages the exchange of information collected along the trajectories investigated by each worker.

This version, PAR-1P, adopts the approach typically used by metaheuristics, where the master keeps the best (or elite) solution currently encountered by any worker. Each time the best solution is improved, the master broadcasts the solution's cost to all workers. The intuition is to use this information not only to converge faster to a target solution, but also to find better solutions than the independent search strategies.

In PAR-1P, there is no one-off cooperation during the first iteration. Instead, each time a worker completes the ILS local search phase, it will compare the cost of the solution found with that of the best solution held by the master. If it is lower, the worker sends its solution to the master, otherwise the solution is discarded. After this synchronization, two outcomes are possible. Either, the worker requests the best solution held by master to repeat the ILS local search phase with this solution, or the worker continues with the next iteration (i.e., constructs a new initial solution during the GRASP phase and proceeds with the next steps of the sequential heuristic) as in the previous versions. The probability of each outcome is denoted by Q and $1 - Q$, respectively. In this way, workers indirectly exchange elite solutions (high-quality solutions) found along their search trajectories.

In this parallel cooperative strategy, synchronization points occurs when a worker sends its best solution to the master and when the worker receives from the master an elite solution. Due to the increased communication in this version, the master does not participate in the search (i.e., it does not execute the GRILS-mTTP heuristic).

5.4. Parallel strategy with a pool of elite solutions

In this parallel cooperative strategy, PAR-nP, the master is dedicated to managing a centralized pool of elite solutions (and their costs), including collecting and distributing them upon request. As in the previous version, workers start their search from different initial solutions and can exchange and share elite solutions found along their search trajectories.

The master will update the elite solution pool with a newly received solution according to given criteria which are based on the quality of the solutions already in the pool (as described below). When a worker completes an iteration, it can either request an elite solution from the pool or construct a new initial solution randomly, again, with probabilities of Q and $1 - Q$, respectively.

Pool management A very important aspect of this algorithm is managing the pool of elite solutions. Empirically, previous research (see e.g. [7]) observed that history-based metaheuristics are less likely to be successful if the recorded solutions are very similar. Therefore, it is necessary to take into account not only solution quality, but also diversity when dealing with pools of elite solutions.

The pool consists of a limited $MaxElite$ number of positions, which are initialized with null solutions. The pool manager supports two essential operations: the insertion of new solutions into their appropriate position in the pool and the selection of a solution from the pool from which a worker will initiate a new search.

To guarantee the diversity within the pool, the insertion of a new solution depends on the state of the pool and how the solution was generated. When the new candidate solution has been derived from an elite solution in the pool, the cost of the new solution must be better than the cost of the elite solution from which is was generated. If true, the new solution will obligatorily take the place of that elite solution. On the other hand, if the solution was derived from a solution produced by the GRASP construction phase, the solution can be inserted directly into any vacant position. In the case where the pool is full, the solution is inserted only if it is as good as the worst elite solution already in the pool (thus replacing that solution).

When a worker process requests an elite solution from the master, a solution is selected at random (using a uniform distribution, which has been applied to other problems with reasonable success [16]) from the pool and sent back to worker.

6. Experimental results

The four parallel algorithms PAR-I, PAR-O, PAR-1P and PAR-nP, described in Section 5, were implemented using C++, the MPI-LAM (version 7.0.4.) implementation of the message passing interface standard MPI and grid-enabled with the EasyGrid AMS middleware [2]. Each processor has a copy of the executable code and the problem data.

Two sets of benchmark instances have been proposed for traveling tournament problems [4]. The first is made up of *circle instances*, artificially generated to represents easier instances. The name circ*n* is used to denote a circle instance with $4 \leq n \leq 20$ teams. Each circle instance is built from a graph, generated as follows. Nodes are placed at equal unit distances along a circumference and are labeled $0, 1, \ldots, n-1$. There are edges only between nodes i and $i+1 \bmod n$, for $i = 0, 1, \ldots, n-1$. In the corresponding circle instance, the distance between the home cities of teams i and j (with $i > j$) is given by the length of the shortest path between them in the graph and is equal to the minimum of $i-j$ and $j-i+n$. The second set are realistic instances generated using the distance between the home cities of a subset of teams playing in the National League of the MLB (Major League Baseball) in the United States. These national league instances are denoted by nl*n* with $4 \leq n \leq 16$. As in [21], this work did not consider the smaller instances

with $n = 4$ and $n = 6$ for which optimal solutions have already been found. Furthermore, an additional real-life instance has been created by Ribeiro and Urrutia [21], named br24. This instance is made up of the home cities of the 24 teams playing in the first division (Série A) of the 2003 edition of the Brazilian soccer championship. All instances and their best known solutions are available from [23].

The experiments aim to investigate how parallel computing can be used to harness cooperation and diversity and thus improve the quality and convergence when executing the GRILS-mTTP metaheuristic in distributed computing environments. The parameter $MaxElite$ was set to P, where P is the number of worker processes used in the parallel execution, and the probability, Q, of choosing a solution from the pool was fixed at 10%.

Table 1 presents the solutions obtained by the sequential and the best parallel implementations for the mTTP, when given at least five days of execution time. For 5 of the 13 benchmark instances, the PAR-nP managed to find better solutions than the sequential GRILS-mTTP. Note that it is still unknown which, if any, are optimal solutions.

Instance	Seq. Soln.	Par. Soln.	Improv.(%)
circ8	140	140	-
circ10	276	272	1.45
circ12	456	456	-
circ14	714	714	-
circ16	1004	980	2.39
circ18	1364	1306	2.39
circ20	1882	1882	-
nl8	41928	41928	-
nl10	63832	63832	-
nl12	120655	120655	-
nl14	208086	208086	-
nl16	285614	279618	2.09
br24	506433	503158	0.65

Table 1. Best solution found in parallel.

In the following experiments, the cost of the best solutions found by the sequential version of GRILS-mTTP (Table 1) is referred to as the *easy* targets.

The improved solutions obtained by PAR-I, PAR-O and PAR-1P parallel implementations for the five instances circ10, circ16, circ18, nl16 and br24, are referred to as the *medium* targets (Table 2). PAR-nP improved three of these instances further (Table 3) which will be referred to as the *hard* targets.

The next experiment analyzes the time taken by the parallel strategies to achieve the medium target, and verify the benefits of exchanging information between the workers instead of letting them execute independently. The results, for a set of $P = 10$ processors, are reported in Table 2. The first column in this table presents the instances for which a new medium target (second column) was found. The following four columns show the overall elapsed time, in seconds, necessary to find the target by the parallel strategies for each instance. The times reported were the best over five runs for each instance.

Instance	Target	PAR-I	PAR-O	PAR-1P	PAR-nP
circ10	272	5590.53	4703.68	8391.18	1802.42
circ16	984	5454.47	1787.18	2490.31	1254.31
circ18	1308	8445.06	27649.50	9952.20	269.23
nl16	280174	1311.47	793.87	8188.81	2205.72
br24	503158	7250.49	7082.72	3082.30	4460.18

Table 2. Computation times in seconds to achieve the medium targets.

Instance	Prev. best	New best	Impr. (%)	Time (s)
circ16	984	980	0.40	783.14
circ18	1308	1306	0.15	25,828.60
nl16	280174	279618	0.20	20,512.50

Table 3. Best solutions found by PAR-nP.

Using a cooperative strategy based on a pool of elite solutions, PAR-nP presents the smallest computation time in most cases. Although PAR-1P also shares information, it only records one elite solution. Therefore, the degree of diversity tends to be smaller than in PAR-nP, possibly leading the workers to search the same region and, consequently, taking longer to converge to the target. In the case of br24, it is likely that more workers concentrate their search around the neighborhood of the elite solution and thus converge faster. Using the same reasoning, with the exception of instance circ18, we see that PAR-O is faster than PAR-I.

Instance	$2 \times t$ (s)	Target	PAR-I	PAR-O	PAR-1P
circ16	2.000	980	990	984	998
circ18	50.000	1306	1308	1308	1384
nl16	40.000	279618	280174	280174	298357

Table 4. Solutions found by PAR-I, PAR-O, and PAR-1P when executed for $2 \times t$ seconds.

Algorithm PAR-nP found better solutions than the other parallel implementations for three of the instances. Table 3 presents in the first column the benchmark instances for which improvements over the medium targets were obtained by PAR-nP, followed by the values of the previous and new best known solutions and the relative percentage improvement. Also, included is the overall elapsed time in seconds that was required to find the new solutions using 10 processors. Note that the solution obtained for the instance

circ18 is also the best known solution for the non-mirrored version of the TTP.

The following experiment aims to investigate the robustness of the cooperative strategies, i.e. how often each version manages to reach the hard target. Compared to the execution times of PAR-nP, which always found the hard targets, we would like to know if PAR-I, PAR-O, and PAR-1P can also manage the same feat. Given t, the time taken by PAR-nP to achieve the best known solution, PAR-I, PAR-O, and PAR-1P were allowed to execute for approximately $2 \times t$ with 10 processors. The best targets found over five executions for each instance are presented in Table 4. For the instances improved by PAR-nP, this table reports the elapsed time allowed to find the given target, the hard target, and the cost of the solutions found by the other parallel implementations.

The scalability of the parallel strategies was also evaluated to study the benefits of searching an increasing number of multiple trajectories. Executing with more processes offers a greater diversity due to the use of multiple distinct seeds. Table 5 shows the average execution times over ten runs, for PAR-I and PAR-O and Table 6, for PAR-1P and PAR-nP. For each instance, both tables show the computation time in seconds required by the sequential version, and the parallel strategies to achieve the easy target using five, ten and twenty processors.

		PAR-I			PAR-O		
Inst.	Seq.	**5**	**10**	**20**	**5**	**10**	**20**
circ8	1.4	1.05	0.21	0.21	0.24	0.13	0.15
circ10	276.0	53.42	9.27	9.28	5.31	22.46	1.80
circ12	8.5	0.10	0.10	0.10	2.20	0.90	0.10
circ14	1.1	0.06	0.06	0.06	0.08	0.09	0.13
circ16	115.3	90.34	29.24	25.30	32.25	42.97	42.02
circ18	284.2	23.64	12.87	6.80	13.93	4.08	4.09
circ20	578.3	101.28	101.28	66.89	454.91	119.73	3.16
nl8	0.7	0.03	0.03	0.03	0.55	0.17	0.06
nl10	643.9	3.73	3.72	3.74	137.18	37.08	37.06
nl2	24.0	2.29	0.40	0.39	3.80	0.23	0.16
nl4	69.9	3.12	3.11	3.12	0.56	0.25	1.83
nl6	514.2	29.33	29.28	29.26	137.13	14.28	14.28
br24	742.5	77.58	77.58	77.82	10.62	11.93	62.19

Table 5. Average computational time for PAR-I and PAR-O on 5, 10 and 20 proc.

For all instances, the parallel versions were faster than the sequential one. As the number of processors available increases, each of the algorithms converges faster. When five processors are used, PAR-I appears to be the best strategy, on average. For 20 processors, PAR-O is better than the rest, on average, in terms of processing time for the easy targets. Remember that PAR-I and PAR-0 benefit from the fact that the master is also involved in the search acting as an additional worker. For most of the instances, mainly for the difficult ones, the cooperative strategies (PAR-O, PAR-1P and PAR-nP) present smaller computation times and scale quite reasonably.

		PAR-1P			PAR-nP		
Inst.	Seq.	**5**	**10**	**20**	**5**	**10**	**20**
circ8	1.4	1.03	0.21	0.21	0.57	0.04	0.04
circ10	276.0	73.86	13.03	9.61	43.64	14.60	14.56
circ12	8.5	0.10	0.10	0.10	2.04	0.32	0.32
circ14	1.1	0.06	0.06	0.06	0.86	0.13	0.13
circ16	115.3	104.54	76.18	17.98	42.51	34.12	4.83
circ18	284.2	14.00	14.00	10.38	27.47	27.48	16.57
circ20	578.3	36.15	138.80	16.09	110.81	18.19	13.99
nl8	0.7	0.03	0.03	0.04	0.09	0.10	0.09
nl10	643.9	50.60	50.43	4.78	139.70	48.22	35.96
nl12	24.0	4.23	0.39	0.39	0.15	0.16	0.15
nl14	69.9	1.43	1.43	1.35	2.34	2.34	2.34
nl16	514.2	406.24	280.90	269.02	477.70	243.16	113.86
br24	742.5	120.11	12.47	12.45	399.62	45.09	7.10

Table 6. Average computational time for PAR-1P and PAR-nP on 5, 10 and 20 proc.

A preliminary investigation into the advantage of searching multiple walks without increasing the number of processors, i.e executing more than one process per processor, revealed that both PAR-O and PAR-nP converge faster when executing ten processes on five processors, rather than just five processes on five processors. This implies that diversity of the seeds helps the algorithms to converge (to the easy targets).

7. Conclusions

Metaheuristics, such as GRASP and ILS, have found their way into the standard toolkit of combinatorial optimization methods. Parallel implementations of metaheuristics are usually applied in the context of hard combinatorial optimization problems often allowing reductions in computational times. Independent strategies can obtain good solutions in terms of solution quality and computation results. However, the parallelization based on cooperative search lead to more robust implementations, which are likely to be the most important contribution of parallelism to metaheuristics.

Compared to their sequential counterparts, parallel metaheuristics demand significantly more programming and design effort. The implementations described in this paper illustrate the strategies and programming skills involved in the development of robust and efficient parallel implementations of metaheuristics.

The results show that the GRILS-mTTP heuristic benefits from parallel implementations, which are capable of finding better solutions with respect to their sequential counterpart. The use of multiple processes and a pool

of elite solutions offers a diversity of high quality solutions from which workers can search for better solutions. The pool also provides a mean to implement cooperation and faster convergence.

A grid enabled version of strategy PAR-nP capable of executing efficiently and robustly in computational grids is currently being evaluated. Results show that this new dynamic grid implementation performs equally as well equivalent static version presented in this paper. This grid version permits the execution for extensive time periods without the user needing to be concerned with resource or process failure and resource utilization. Ongoing work is investigating combinations of heuristic strategies and parameter settings appropriate for cluster and grid environments.

References

[1] A. Anagnostopoulos, L. Michel, P. V. Hentenryck, and Y. Vergados. A simulated annealing approach to the traveling tournament problem. In *Proceedings of CPAIOR'03*, 2003.

[2] C. Boeres and V. Rebello. Easygrid: Towards a framework for the automatic grid enabling of legacy mpi applications. *Concurrency And Computation Practice And Experience*, 17(2):173–190, 2004.

[3] V.-D. Cung, S. Martins, C. Ribeiro, and C. Roucairol. Strategies for the parallel implementation of metaheuristics. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 263–308. Kluwer, 2002.

[4] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem: Description and benchmarks. *Lecture Notes in Computer Science*, 2239:580–584, 2001.

[5] K. Easton, G. Nemhauser, and M. Trick. Solving the travelling tournament problem: a combined integer programming and constraint programming approach. In *Lecture Notes in Computer Science*, volume 2740, pages 100–109, 2003.

[6] S. Elmohamed, P. Coddington, and G. Fox. A comparison of annealing techniques for academic course scheduling. *Lecture Notes in Computer Science*, 1408:92–114, 1998.

[7] E. Fernandes and C. Ribeiro. Using an adaptive memory strategy to improve a multistart heuristic for sequencing by hybridization. *Lecture Notes in Computer Science*, 3503:4–15, 2005.

[8] P. Festa and M. Resende. GRASP: An annotated bibliography. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 325–367. Kluwer, 2002.

[9] I. Foster. *Designing and Building Parallel Programs*. Addison-Wesley, 1995.

[10] I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. 2nd edition. Morgan Kaufmann, 2004.

[11] P. Hansen and N. Mladenovic. Developments of variable neighborhood search. In C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer, 2002.

[12] M. Henz. Scheduling a major college basketball conference revisited. *Operations Research*, 49:163–168, 2001.

[13] H. Lourenco, O. Martins, and T. Stutzle. Iterated local search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer, 2002.

[14] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

[15] G. Nemhauser and M. Trick. Scheduling a major college basketball conference. *Operations Research*, 46:1–8, 1998.

[16] M. Resende and C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer, 2003.

[17] M. Resende and C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Kluwer, 2005.

[18] C. Ribeiro and I. Rosseti. A parallel GRASP for the 2-path network design problem. *Lecture Notes in Computer Science*, 2400:922–926, 2002.

[19] C. Ribeiro and I. Rosseti. Efficient parallel cooperative implementations of GRASP heuristics, 2005. Submitted for publication.

[20] C. Ribeiro, R. Souza, and C. Vieira. A comparative computational study of random number generators. *Pacific Journal of Optimization*, to appear.

[21] C. Ribeiro and S. Urrutia. Heuristics for the mirrored traveling tournament problem. *European Journal of Operational Research*, to appear.

[22] J. Thompson. Kicking timetabling problems into touch. *OR Insight*, 12:7–15, 1999.

[23] M. Trick. Challenge traveling tournament instances. Online document at http://mat.gsia.cmu.edu/TOURN/ , last visited on May 29, 2005.

[24] M. Verhoeven and E. Aarts. Parallel local search. *Journal of Heuristics*, 1:43–65, 1995.

[25] M. Wright. Scheduling English cricket umpires. *Journal of the Operational Research Society*, 42:447–452, 1991.

[26] J. Yang, H. Huang, and J. Horng. Devising a cost effective basketball scheduling by evolutionary algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 1660–1665, Honolulu, 2002.