Parallel Tabu Search and the Multiobjective Vehicle Routing Problem with Time Windows

Andreas Beham

***Abstract*—In this paper the author presents three approaches to parallel Tabu Search, applied to several instances of the Capacitated Vehicle Routing Problem with Time Windows (CVRPTW). Attention in this work was given to keep the parallel implementations simple. The parallel algorithms are of two kinds: Two of them are parallel with respect to functional decomposition and one approach is a collaborative multisearch TS. The implementation builds upon a framework called Distributed metaheuristics or DEME for short. Tests were performed on an SGI Origin 3800 supercomputer at the Johannes Kepler University of Linz, Austria.**

***Index Terms*—parallel, multiobjective, tabu search, vehicle routing**

I. Introduction

Tabu Search was invented by Fred Glover [1] and has since then appeared in numerous scientific papers and was target of many investigations. Tabu Search is basically a "best-improvement-local-search" algorithm that uses one solution to generate a number of moves leading to solutions that are considered to be neighbors of the current solution. From this neighborhood Tabu Search will choose the best solution and continue by creating a new neighborhood from there. To avoid undoing changes made in previous moves, Tabu Search stores recent moves in the *tabulist*. It then forbids to make moves towards a configuration that it had already visited before. Over the years Tabu Search became a metaheuristic well known for its good performance when applied to the Vehicle Routing Problem (VRP) [2] [3] [4].

The VRP is a generalized name for a class of problems first formulated in the late 50s by Dantzig and Ramser [5]. The basic layout of the problem consists of a depot that houses several vehicles and a number of customers scattered or clustered around the depot that need to be serviced by the vehicles. The simplest version of the VRP is probably the Capacitated VRP (CVRP) where each vehicle has a fixed maximum capacity. The length of a tour is thus limited by the demand of the customers and the capacity in the vehicle. In another popular version of the VRP that builds on the CVRP, each vehicle on a tour must arrive in a specified Time Window (VRPTW) at the customer. This includes a lower bound (ready time), an upper bound (due date) and a time value for the duration of the delivery process (service time). The VRPTW can then be further categorized in a formulation with *hard Time Windows* and *soft Time Windows* respectively. In the definition of hard Time Windows, a solution is feasible if and only if each customer is reached before his due date. Contrary to soft Time Windows, where the time window constraints are relaxed and the question of feasibility is left to the designer. Other versions of the VRP include the Multiple Depots VRP (MDVRP) or the VRP with Pickup and Delivery (VRPPD) where customers receive and return goods from and to the depot. There are still many more and combinations of these constraints are common. Together they create a multitude of NP-hard optimization problems.

Parallel Tabu Search has been and continues to be a very active topic [6] and several strategies have been tried focusing on different levels of parallelization, i.e. functional decomposition, domain decomposition and multisearch. In the presented approaches two of them make use of functional decomposition and while a synchronous parallelization was applied early already [7], asynchronous algorithms are not yet common at this level of parallelization. Yet the asynchronous algorithms are interesting as they should perform well on both homogenous and heterogenous systems. We will see that an asynchronous master-worker algorithm is capable of reaching good solutions and that runtime performance can be further increased.

Domain decomposition was introduced to Tabu Search in a concept known as "Adaptive Memory". Adaptive memory is represented as a pool of solution parts from which new solutions are created. During the search good parts are identified and added to the memory. For example, Taillard et al. use a Tabu Search heuristic with an adaptive memory to solve the CVRPsTW in [8]. In [9] the adaptive memory and thus the decomposition was parallelized in a hierarchical way with a managing process, several decomposing processes, a dispatcher and a number of tabu searchers that solve subproblems selected by the decomposers. This is however just an abstraction and was realized on one machine organizing the search and P processors applying heuristic and metaheuristic techniques on the problem or on parts of it. Separating search and strategy is useful when working with large parallel environments.

Multisearch spans from trivial implementations with shared memories to very complex approaches that include strategic knowledge. The third approach presented here works at this level. In this work it was chosen to keep with a simple implementation and exchange new best solutions within other searchers. We will see that this increases runtime in comparison to the sequential algorithm, but leads to improved and more robust solutions. A combination of multisearch and functional decomposition could combine the best of two worlds.

The paper is further organized as follows: In section II the problem will be presented briefly, along with a num-

Andreas Beham is with the Institute for Formal Models and Verification, Johannes Kepler University, Altenberger Str. 69, 4040 Linz, Austria. E-mail: andreas@heuristiclab.com

ber of operators that our algorithms use to transform the problem. Additionally, it will be discussed how a multiobjective formulation does make sense in the case of a real world application. In section III we will discuss the approaches and their implementation. Results are given in section IV and conclusions are drawn in section V.

II. The Capacitated Vehicle Routing Problem with Time Windows

The problem is defined as a set of customers that are to be visited exactly once and by one vehicle. The vehicles, denoted by set V, visit at least one customer and return to the depot. Not all vehicles in V are used though and some may stay at the depot visiting no customers. In fact, one goal of the optimization process is to use as little vehicles as possible. The maximum number of available vehicles at the depot is given by R. We thus have the set $S = \{0, .., N\}$ representing the indices of all *sites* or *locations* involved in the problem. The index 0 is reserved for the depot, thus the set of customers $C = \{1, .., N\}$ is represented by indices ranging from one to N.

Each site s_i is connected to every other site s_j with $i \neq j$ and an associated travel costs $t_{i,j} \in S \times S$ given in matrix T. This matrix is computed by calculating the Euclidean distance between the location's x and y coordinates. Additionally each customer has a demand d_i that represents the amount of goods to deliver, likewise each vehicle has a maximum carrying capacity m. Because the fleet is assumed to be homogeneous this capacity is the same for all vehicles and the index was omitted.

Customers are able to service a vehicle within a specified time frame given as ready time a_i and due date b_i. Once a vehicle arrives at a customer it is delayed by a service time c_i. If a vehicle arrives before the ready time of a customer it has to wait until the customer is ready. In contrast, arriving after the due date constitutes a constraint violation. In this work the problem with soft time windows is considered and the constraint violation is part of the function to be optimized.

A. Representation

A permutation coding was used to represent a solution. Each tour starts and ends at the depot, thus the index 0 is the first and last character of each permutation string. All the tours are concatenated together in one string, removing consecutive zeros. For each vehicle for which no tour was assigned, a 0 is appended to the string. The maximum number of vehicles R is a parameter specified for every instance of the problem and ranges from 25 for the 100 city problems up to 100 for the 400 city problems. The length of the final permutation $\mathcal{P}$ is denoted by $L = |\mathcal{P}| = N + R + 1$ An example permutation string for a problem with 4 customers, one depot and 5 vehicles thus could be: $\mathcal{P} = (0, 4, 2, 0, 3, 0, 1, 0, 0, 0)$

To evaluate the quality of the solution, three objectives are used. The first objective is the optimization of the total tour length.

$$f_1 : \mathcal{N}_0^L \rightarrow \mathcal{R}, \sum_{i=1}^{N-1} t_{i,i+1}$$

The purpose is to ensure that the vehicles move from customer to customer most efficiently, i.e. on the shortest possible path. The second goal is to minimize the number of vehicles that are actually deployed for the scenario. This *may* be in contradiction to the first goal, i.e. when the relation $t_{i,k} + t_{k,j} \geq t_{i,j}, i \neq k \neq j$ does not hold. In this work however the Euclidean distance between the customers was used. From this it follows that a reduction in the number of vehicles from solution a to solution b also results in a total tour length that is less or equal to the other: $f_1(a) \geq f_1(b)$. So in Euclidean space the number of vehicles is minimized with the minimization of the total tour length. Nevertheless the situation remains that solutions with different number of vehicles can have the same tour length and aiming for the second goal thus minimizes the number of vehicles deployed. Considering the representation this objective is the number of times a 0 is followed by a non-zero value.

$$f_2 : \mathcal{N}_0^L \rightarrow \mathcal{N}, ||\{i \in \mathcal{P} | s_i = 0 \wedge s_{i+1} > 0\}||$$

The third objective is to minimize the constraint violation. Constraint violation can occur when the vehicle arrives at customer i after its due date b_i. It is also possible that the carrying capacity m of a certain vehicle becomes smaller than the sum of the demands d of all customers serviced by this vehicle, but because of the design of the operators, this violation could not occur in this example. The third goal is thus a function that calculates and sums the total *tardiness*.

$$f_3 : \mathcal{N}_0^L \rightarrow \mathcal{R}, \sum_{i=0}^{L} Max(ArrivalTime(s_i) - b_{s_i}, 0)$$

Generally allowing solutions with constraint violations in the search trajectory hands more freedom to the algorithm and the moves it can consider at each iteration, however when the violations become too large the solution becomes less and less interesting. While it is acceptable that the algorithm moves outside of the boundaries for a time, it is desirable that it returns to the space of feasible solutions.

B. Operators

Several operators have been presented in the literature and many approaches made good use of them [4]. Of these operators five were selected, giving each operator the same chance to create a neighboring solution. The operators in brief are called: Relocate, Exchange, 2-opt, 2-opt* and or-opt. Relocate moves a customer from one route to another, Exchange swaps two customers of different routes. Relocate and Exchange are thus similar to a $(1, 0)$ and $(1, 1)$ λ-Exchange defined by [10]. 2-opt reverses a tour or a part of it, whereas 2-opt* interchanges 2 tours by crossing the first half of one tour with the second half of another and

vice versa. Last but not least, or-opt moves two consecutive customers to a different place in the same tour.

Additionally a local feasibility criterion was added to each operator. This criterion disallowed to manipulate a solution when it would obviously violate the time window constraints on a local level. In the example of the Relocate operator, it was not allowed to insert a customer k between two other customers i and j, if either $a_i + c_i + t_{i,k} > b_k$ or $a_k + c_k + t_{k,j} > b_j$ were satisfied or the demand of that route exceeds m. This criterion was weak enough that solutions with time window violations occur and strong enough, that the algorithm could find back to a solution with all time windows satisfied.

C. Multiobjective Approach

While multiobjective or multi criteria problems are several times more complex than singleobjective problems, the advantage is to present a choice to the person who is applying the results in the real world. Assuming this work would have practical relevance to a certain customer, instead of handing him one solution with a given tour and a number of vehicles, we may have found solutions with different travel distances and different numbers of vehicles. The customer whose fleet and tours are to be optimized can then decide, based on concrete solutions, which of them is most suitable for his or her business. Solving the problem a number of times with modified weights and a single criteria approach can result in several pareto-optimal solutions as well, however if weights are to be selected randomly the additional effort of MO optimization may shrink considerably against the additional computational effort of the single criteria approach. Alternatively it is possible to calculate a small number of weights through analytical research on the customer's situation. However this requires information which may or may not be existent and could require the customer to either disclose sensible information to a third party or to spend time and money to gather the data. Both situations may be undesirable and may provide a barrier to apply the optimization as a whole. Doing an unbiased search transforms the huge space of possibilities into several concrete solutions which help the customer to attain a decision.

III. Parallel Tabu Search

A. Multiobjective Optimization and Tabu Search

Before we start to discuss the parallel approaches, let us take a quick look on Tabu Search and Multiobjective Optimization (MO). So far research around MO concentrated on evolutionary and population based methods and resulted in several algorithms such as NSGA-II [11], SPEA2 [12], PAES [13] or even multiobjective Scatter Search like MOSS [14] and AbYSS [15] where version numbers in two of them show that they have already been improved. Trajectory based methods such as Simulated Annealing, Variable Neighborhood Search or Tabu Search, to name but a few, attracted less interest. An investigation of Tabu Search algorithm for MO optimization resulted in the MOTS algorithm which was presented by Hansen in [16], however a search on Google Scholar showed only 53 citations for Hansen's paper and well beyond 200 or even 300 citations for the SPEA2 and NSGA-II algorithms.

In this work, the Tabu Search algorithm used does not differ much from the original Tabu Search presented by Glover, though it builds upon what has emerged in multiobjective EAs, mainly the pareto concept to store non-dominated solutions in a memory and the use of an archive to store the non-dominated front that has been found so far. However the focus in this work is not on evaluating the quality of its results against those of other algorithms, but to give a comparison of several parallel approaches. Evaluating the quality of the obtained results and comparing them to leading MO algorithms, such as those mentioned above, might be an interesting topic for a later paper.

B. Sequential Tabu Search

The sequential algorithm is outlined in Algorithm 1. This will be the base from which to develop the parallel algorithms. The algorithm uses three memories, the first memory is the *tabu list* and common to all Tabu Search algorithms. The tabu list is organized as a queue and will hold information about the moves made. When the tabu list is full it will forget about the oldest moves. The length of the tabu list can be specified by the *tabu tenure* parameter and because every iteration there is only one move made this is also the number of iterations the solutions will stay in the tabu list. The second memory is the *medium-term memory* $\mathcal{M}_{nondom}$ that keeps a list of non-dominated solutions that had been found in past neighborhoods. If the algorithm does not find better solutions for a certain number of iterations, it will attempt to try one of the solutions from this memory instead of generating a new neighborhood from the current solution. Better solutions are solutions that dominate the current pareto front, which is stored in the third memory $\mathcal{M}_{archive}$, or that are non-dominated to the front and can be added to $\mathcal{M}_{archive}$. A chosen solution can be added to the archive when it is not dominated to the solutions in the archive and when the archive is not full. If the archive is full, the solution is added based on the result of a *crowding comparison*[11]. This comparison orders the solutions in the archive and the chosen solution by a *distance value*, which is computed by calculating the differences of the fitness values of a certain solution with respect to the other solutions. A solution that has a low distance value has similar fitness values compared to the rest of the solutions and will be deleted. This ensures that the solutions will be spread over the pareto front more equally instead of clustering at a certain position.

The algorithm starts by generating an initial solution, specifically to the CVRPTW the I1-heuristic [17] with randomly chosen parameters was used. The I1-heuristic is a route construction heuristic for the VRP. starts with either the customer with the earliest deadline or the one farthest away, this parameter was controlled randomly. It adds customers based on a savings value that computes the ad-

ditional distance as well as time windows that the insertion of a customer will cost. It will select the customer whose savings value is minimal and insert him at the appropriate position. Afterwards the memories are initialized and the algorithm enters the main loop, which consists of: Neighborhood Generation, Evaluation and a Selection of one of the non-dominated solutions found. If no solution could be selected, due to the tabu criterion or if there has been no improvement to the archive of non-dominated solutions found, then a new current solution is chosen from the memory. In the last step the memories are updated, additional non-dominated solutions that were found in the neighborhood $\mathcal{N}$ are stored in the memory termed $\mathcal{M}_{nondom}$, the chosen current solution is added to the archive $\mathcal{M}_{archive}$ if it is non-dominated to solutions already in the archive or if it dominates them.

The Neighborhood Generation draws a number of moves, specified in the neighborhood size parameter from the five operators described in II.B. For each move to create one of the operators is chosen at random, with equal probabilities for each. If the operator was unable to find a suitable move, with regard to the local feasibility criterion, a new random number is drawn and possibly a different operator is selected. This step is repeated until the amount of moves matches the neighborhood size.

Algorithm 1 The sequential TSMO-Algorithm

```
1:  procedure TSMO
2:      s ← GenerateInitialSolution()
3:      evaluations ← 0, iterations ← 0
4:      M ← InitializeMemories()
5:      while evaluations < MaximumEvaluations do
6:          N ← GenerateNeighborhood(s ↓)
7:          Evaluate(N ↓↑, evaluations ↓↑)
8:          s ← Select(N ↓, M_tabulist ↓)
9:          if s ∉ N ∨ noImprovement then
10:             s ← SelectFrom(M_nondom ↓↑ ∪ M_archive ↓)
11:             noImprovement ← false
12:         end if
13:         M ← UpdateMemories(s ↓, N ↓)
14:         if isUnchanged(M_archive ↓, iterations ↓) then
15:             noImprovement ← true
16:         end if
17:         iterations ← iterations + 1
18:     end while
19: end procedure
```

C. Synchronous TS

The first parallel approach is a very simple parallelization of the GenerateNeighborhood() and Evaluate() functions using a master process that distributes the work among himself and several worker processes. This approach does not improve the quality of the results, but the runtime. It is synchronized in that the master selects the current individual, distributes the work and waits to collect all the results. Sometimes it is reasonable to just parallelize the evaluation function, i.e. when the neighborhood generation is more primitive. But in the example of the VRP and our operators coupled with the local feasibility criterion it made sense to parallelize the generation as well, as it consumed a large part of the time. The master thus sends to each worker the current individual and the number of neighbors to generate. The workers will then do their calculation and send their part of the neighborhood back to the master. When all neighbors are collected the master continues with the selection and the rest of the iteration.

While we should expect to get an improvement in runtime the drawback with this approach is that the processors wait a considerable amount of time. The advantage is that this algorithm can be compared easily to the sequential algorithm as the behavior remains unchanged.

D. Asynchronous TS

In this approach we try to minimize the waiting time and thus the disadvantage of the synchronous version. Unfortunately this is not possible without changing the behavior of the algorithm and so we expect to see different results. The asynchronous TS still uses a master-worker philosophy and parallelizes the neighborhood generation and evaluation function, but the master does not wait in all cases for the workers to continue. Like the synchronous algorithm it will distribute the work among himself and the workers, but when it is finished with its part, the master will use a decision function to decide if workers should be given more time or if it should continue by selecting the next current individual from the $\mathcal{N}$ that has been collected so far. Thus the master can consider only parts of a neighborhood per iteration and will take the other parts into account once they will be evaluated. The search can select solutions that were neighbors of a previous solution, but not evaluated at the time the algorithm continued from the previous to the next. It could also select solutions that had been part of the same neighborhood that the current solution was selected from. The behavior is illustrated in Fig. 1. We will see in section IV that this approach is quite faster than the synchronous TS and of similar quality, based on the evaluation of 100,000 solutions. To decide wether the master should wait for the slaves to finish their evaluations or continue selecting an individual from those moves that had already been evaluated a decision function is used. This function has several conditions. The master will stop waiting for unfinished evaluations if either of these conditions satisfy. The function is shown in Algorithm 2.

Algorithm 2 Decision function of the asynchronous TS

```
1: procedure DECISION(current ↓ N ↓ workers ↓)
2:     c1 ← {w ∈ workers | w = waiting}
3:     c2 ← {s ∈ N | s dominates current}
4:     c3 ← AreWeWaitingTooLong()
5:     c4 ← evaluations ≥ MaximumEvaluations
6:     return |c1| > 0 ∨ |c2| > 0 ∨ c3 ∨ c4
7: end procedure
```

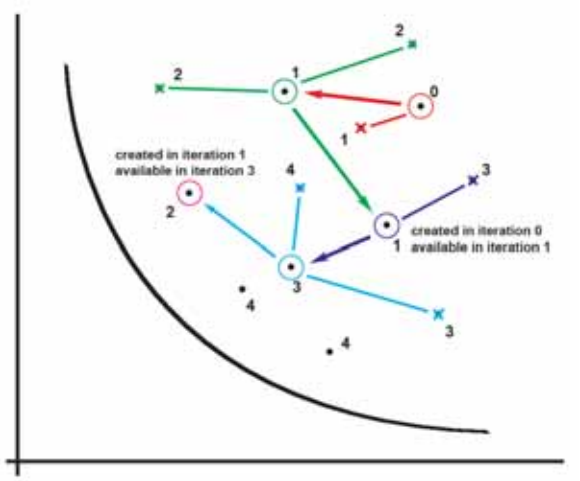

Fig. 1. A fictional search trajectory for the asynchronous TS approaching the pareto-optimal front. The numbers denote the iteration at which the solution was created. Equal numbers denote solutions belonging to the same neighborhood. The circles mark solutions which have been selected as current solutions. From these the new neighborhoods were created. The arrows show the path of the trajectory and the other lines mark those solutions which had been considered as possible new current solutions during a given iteration.

E. Multisearch TS

The third approach is asynchronous and is placed in the realm of multisearch parallel algorithms. The parameters of the algorithm for each, but the first, are disturbed by a random variable derived from a normal distribution with mean 0 and a standard deviation that is the quarter of the parameter to be disturbed. The algorithms then work in a similar way to the sequential algorithm, but after an initial phase they communicate improving solutions that they found along the pareto front. An improving solution here is a solution that could be added to the pareto front which is stored in $\mathcal{M}_{archive}$. Likewise a non-improving solution is one that could not be added, because it is dominated or too crowded. The initial phase starts with the beginning of the search and ends when the algorithm could not add any new solutions to the set of pareto optimal solutions found for a number of iterations. This means that the algorithm has found an initial set of good solutions, and has finally made a number of non-improving moves. The communication list is initialized randomly before the main loop and different for every process. It is used to determine the process that will receive the next improving solution found.

When one process finds an improving solution it is likely that this solution is of interest to other processes as well. However to keep the communication overhead small and to prevent all processes from searching the same region, the process that finds an improving solution will send it to a single other process only. The process which to send it to is determined by the first place in the communication list. After the solution has been sent the communication list is rotated in that the first process is moved to the bottom. The process receiving the individual tries to store the solution in its memory of non-dominated solutions $\mathcal{M}_{nondom}$. It is added if it is non-dominated or dominates the solutions there and otherwise discarded. This way the communication overhead does not become too large and good solutions find their way to other searchers who can explore this region as well, if they do not find improving solutions in the region they currently are. This rather simple approach already leads to improved solutions.

IV. Results

The empirical results on a number of test problems are shown through Tables I to IV. The results were computed on an SGI Origin 3800 supercomputer located at the Johannes Kepler University, Linz, Austria. The Origin 3800 is a shared memory machine with 64GB RAM and consists of 128 R12000 MIPS processors, running at 400Mhz each.

Tables I and II show the averaged results from the 400 city extended Solomon problems, where Table I shows results from the problems with small time windows and Table II shows results from the problems with large time windows respectively. The problemset is available at a website managed by Joerg Homberger[1]. Tables III and IV show results from the 600 city problems of the same problemset.

While during the run of the algorithms solutions with constraint violations had been allowed, these solutions were excluded for the generation of the results. Only those solutions were considered that did not violate the time-window and capacity constraints. The first and second columns show the result of the two goals distance and number of vehicles. Average runtime in seconds is shown in the third column, the fourth column lists results from the set coverage metric [18]. This metric measures the ratio between dominated and total solutions of one algorithm against the solutions found by another. The first value shows the percentage of solutions found by one algorithm that dominate those found by the other algorithms, whereas the second value shows the percentage of domination of the other algorithms compared to the one we are looking at. A value of 100% means that the algorithm in question dominates all the solutions found by the other algorithms, so the higher the number the better the quality with respect to the other algorithms. The metric is computed by comparing each run of a problem with all runs of another algorithm for that same problem and averaging the result. The final score is the average of all runs of all problems compared against all runs of all problems of all other algorithms.

As expected the synchronous algorithm completes faster than the sequential algorithm, its solution qualities however are not improving. This is not surprising as the behavior of the synchronous algorithm does not differ from the sequential one. Also a maximum speedup seemed to be reached quickly with a few number of processors already. The formula to calculate the average speedup value is $speedup = T_s/T_p$, the mean execution time of the sequential algorithm divided by the mean execution time of the parallel algorithm.

[1] http://www.fernuni-hagen.de/WINF/touren/inhalte/probinst.htm

The asynchronous algorithm is even faster and pleasingly obtains results that are comparable to those found by the synchronous and the sequential TS. It can also benefit more from an increasing number of processors and achieves by far the best speedup. Though the communication overhead becomes noticeable at 12 processors when the speedup is decreasing from the value it obtained at 6 processors. Given an equal amount of time, it would be possible for the asynchronous Tabu Search to do more evaluations, which would amount for another interesting comparison.

The collaborative TS performed very well when examining the solution qualities. It did better than the two other algorithms, especially at finding solutions that use less vehicles. The runtime performance is worse as it does not share the work in a way the synchronous and asynchronous TS do. Essential it performs a sequential algorithm with communication between the processors. The increasing runtime values thus reflect the time the processors use for communicating with each other. A combination of this collaborative tabu search and the asynchronous TS would make sense to exploit even more processors and may be an interesting future research topic.

To test the statistical significance a pairwise t-test was performed on the results. In the case with 3 processors a 5% significance level could not be achieved all the time for the collaborative TS. The p-values range between 0.1033 and 0.0318 with an average of 0.0599. Using 6 and 12 processors the p-value for the collaborative TS stayed below the 5% significance level in all cases. The results of the master slave and the sequential algorithms do not show a significant difference. Their p-values range between 0.1664 and 0.2541 with an average of 0.2120.

V. Conclusions & Future Work

We have seen that improving performance in both runtime and solution quality does not require a lot of effort and where parallel systems are available researchers should make use of the extra processing power. An interesting approach is the asynchronous master slave algorithm as it performed quite well, yet achieved a much better speedup than the synchronized algorithm. Wether the collaborative TS in this simple form delivers a good tradeoff between extra solution quality and runtime performance is questionable, however the coverage metric suggests that solutions found were more robust than just the master slave versions of the algorithm.

What remains for the future would be a comparison between the TSMO versions here and the well established multiobjective evolutionary algorithms in both runtime and solution quality and on different problems, as well as combining the multisearch TS with the asynchronous TS to get the best of both worlds and probably an algorithm that delivers both good solutions and runtime performance.

References

[1] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research*, vol. 13, pp. 533–549, 1986.

[2] J.-F. Cordeau and G. Laporte, "Tabu search heuristics for the vehicle routing problem," University of Montreal, Canada, Technical Report G-2002-15, 2002.

[3] O. Bräysy and M. Gendreau, "Tabu search heuristics for the vehicle routing problem with time windows," SINTEF Applied Mathematics, Department of Optimisation, Norway, Technical Report STF42 A01022, 2001.

[4] O. Bräysy and M. Gendreau, "Vehicle routing problem with time windows, Part ii: Metaheuristics," *Transportation Science*, vol. 39, no. 1, pp. 119–139, 2005.

[5] G. B. Dantzig and R. Ramser, "The Truck Dispatching Problem," *Management Science*, vol. 6, p. 8091, 1959.

[6] T. Crainic, M. Gendreau, and J. Potvin, "Parallel tabu search," in *Parallel Metaheuristics: A New Class of Algorithms*, E. Alba, Ed. John Wiley & Sons, September 2005, pp. 289–314.

[7] B. Garcia, J. Potvin, and J. Rousseau, "A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints," *Computers & Operations Research*, vol. 21, no. 9, pp. 1025–1033, 1994.

[8] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J. Potvin, "A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows," *Transportation Science*, vol. 31, no. 2, pp. 170–186, 1997.

[9] P. Badeau, M. Gendreau, F. Guertin, J. Potvin, and E. Taillard, "A Parallel Tabu Search Heuristic For The Vehicle Routing Problem With Time Windows," *Transportation Research C*, vol. 5, no. 2, pp. 109–122, 1997.

[10] I. Osman, "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem," *Annals of Operations Research*, vol. 41, no. 1–4, pp. 421–451, 1993.

[11] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan, "A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II," in *Proceedings of the Parallel Problem Solving from Nature VI Conference*, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds. Paris, France: Springer. Lecture Notes in Computer Science No. 1917, 2000, pp. 849–858. [Online]. Available: citeseer.ist.psu.edu/deb00fast.html

[12] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," Gloriastrasse 35, CH-8092 Zurich, Switzerland, Tech. Rep. 103, 2001. [Online]. Available: citeseer.ist.psu.edu/article/zitzler01spea.html

[13] D. W. Corne, J. D. Knowles, and M. J. Oates, "The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization," in *Proceedings of the Parallel Problem Solving from Nature VI Conference*, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, Eds. Paris, France: Springer. Lecture Notes in Computer Science No. 1917, 2000, pp. 839–848. [Online]. Available: citeseer.ist.psu.edu/corne00pareto.html

[14] R. Beausoleil, "MOSS - Multiobjective Scatter Search Applied to Nonlinear Multiple Criteria Optimization," *European Journal of Operational Research*, vol. 169, no. 2, pp. 426–449, 2006.

[15] A. J. Nebro, F. Luna, E. Alba, A. Beham, and B. Dorronsoro, "AbYSS: Adapting Scatter Search for Multiobjective Optimization," Dpto. de Lenguajes y CC.CC., Universidad de Málaga, Technical Report ITI 06-02, 2006.

[16] M. P. Hansen, "Tabu Search in Multiobjective Optimisation : MOTS," in *Proceedings of the 13th International Conference on Multiple Criteria Decision Making (MCDM'97)*, Cape Town, South Africa, 1997, pp. 574–586. [Online]. Available: citeseer.ist.psu.edu/hansen97tabu.html

[17] M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints," *Operations Research*, vol. 35, pp. 254–265, 1987.

[18] E. Zitzler, "Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications," Doctoral thesis, ETH No:13398, Swiss Federal Institute of Technology, Zurich, Switzerland, 1999.

Algorithm	distance	vehicles	runtime	coverage	speedup
Sequential TSMO	$226897.72_{\pm 4999.31}$	$936.59_{\pm 27.20}$	$2226.33_{\pm 80.10}$		
3 cpus					
TSMO sync.	$227539.44_{\pm 5280.69}$	$944.54_{\pm 28.28}$	$1958.97_{\pm 63.85}$	$24.45\% \leftrightarrow 41.22\%$	13.65%
TSMO async.	$221850.65_{\pm 4352.95}$	$919.56_{\pm 23.64}$	$1105.77_{\pm 60.16}$	$31.29\% \leftrightarrow 34.16\%$	101.34%
TSMO coll.	$215624.70_{\pm 3613.35}$	$881.11_{\pm 16.81}$	$2626.53_{\pm 112.88}$	$50.35\% \leftrightarrow 15.99\%$	−15.24%
6 processors					
TSMO sync.	$225873.44_{\pm 4479.61}$	$942.03_{\pm 26.53}$	$1851.80_{\pm 79.41}$	$23.06\% \leftrightarrow 43.34\%$	20.23%
TSMO async.	$222838.20_{\pm 4209.82}$	$914.94_{\pm 22.86}$	$878.77_{\pm 42.05}$	$28.06\% \leftrightarrow 36.90\%$	153.35%
TSMO coll.	$211201.93_{\pm 3187.96}$	$858.99_{\pm 13.19}$	$2813.27_{\pm 85.13}$	$58.26\% \leftrightarrow 9.01\%$	−20.86%
12 processors					
TSMO sync.	$225680.59_{\pm 4532.61}$	$935.14_{\pm 25.77}$	$1802.17_{\pm 52.98}$	$22.01\% \leftrightarrow 45.29\%$	23.54%
TSMO async.	$222831.96_{\pm 4296.43}$	$929.01_{\pm 25.21}$	$1228.06_{\pm 54.10}$	$25.03\% \leftrightarrow 42.33\%$	81.29%
TSMO coll.	$206091.27_{\pm 2679.83}$	$842.76_{\pm 10.86}$	$3056.13_{\pm 200.50}$	$67.39\% \leftrightarrow 4.77\%$	−27.15%

TABLE I

RESULTS ON THE 400 CITY EXTENDED SOLOMON PROBLEMS WITH SMALL TIME WINDOWS (C1, R1). EACH PROBLEM WAS RUN 30 TIMES, THE MAXIMUM NUMBER OF EVALUATIONS WAS SET TO 100,000, NEIGHBORHOOD SIZE WAS SET TO 200 AND IF NO BETTER SOLUTION WAS FOUND AFTER 100 ITERATIONS, A RESTART WITH AN INDIVIDUAL FROM THE MEMORY WAS ATTEMPTED. THE SIZE OF THE ARCHIVE WAS SET TO 20 AS WAS THE VALUE OF THE TABU TENURE

Algorithm	distance	vehicles	runtime	coverage	speedup
Sequential TSMO	$177541.24_{\pm 4147.06}$	$434.15_{\pm 41.19}$	$2794.97_{\pm 100.50}$		
3 processors					
TSMO sync.	$177176.40_{\pm 4123.30}$	$432.60_{\pm 41.55}$	$2444.00_{\pm 63.69}$	$25.83\% \leftrightarrow 36.37\%$	14.36%
TSMO async.	$175137.21_{\pm 4061.77}$	$412.34_{\pm 37.14}$	$1618.00_{\pm 59.28}$	$27.24\% \leftrightarrow 34.67\%$	72.74%
TSMO coll.	$168620.68_{\pm 3281.55}$	$347.45_{\pm 25.12}$	$3213.23_{\pm 88.06}$	$47.08\% \leftrightarrow 15.04\%$	−13.02%
6 processors					
TSMO sync.	$175901.49_{\pm 4273.42}$	$418.76_{\pm 36.45}$	$2307.10_{\pm 63.61}$	$23.77\% \leftrightarrow 38.07\%$	21.15%
TSMO async.	$175142.00_{\pm 3841.85}$	$417.11_{\pm 38.06}$	$1286.60_{\pm 55.95}$	$24.92\% \leftrightarrow 38.02\%$	117.24%
TSMO coll.	$164841.45_{\pm 2827.13}$	$314.84_{\pm 19.28}$	$3425.30_{\pm 108.69}$	$55.56\% \leftrightarrow 7.96\%$	−18.40%
12 processors					
TSMO sync.	$176938.73_{\pm 4309.49}$	$417.35_{\pm 38.41}$	$2313.53_{\pm 66.17}$	$22.24\% \leftrightarrow 42.35\%$	20.81%
TSMO async.	$176167.42_{\pm 4294.61}$	$413.54_{\pm 38.72}$	$1353.13_{\pm 48.72}$	$24.20\% \leftrightarrow 40.47\%$	106.56%
TSMO coll.	$161061.75_{\pm 2736.40}$	$292.67_{\pm 16.07}$	$3613.87_{\pm 102.24}$	$63.67\% \leftrightarrow 4.09\%$	−22.66%

TABLE II

RESULTS ON THE 400 CITY EXTENDED SOLOMON PROBLEMS WITH SMALL TIME WINDOWS (C2, R2). EACH PROBLEM WAS RUN 30 TIMES, THE MAXIMUM NUMBER OF EVALUATIONS WAS SET TO 100,000, NEIGHBORHOOD SIZE WAS SET TO 200 AND IF NO BETTER SOLUTION WAS FOUND AFTER 100 ITERATIONS, A RESTART WITH AN INDIVIDUAL FROM THE MEMORY WAS ATTEMPTED. THE SIZE OF THE ARCHIVE WAS SET TO 20 AS WAS THE VALUE OF THE TABU TENURE

Algorithm	distance	vehicles	runtime	coverage	speedup
Sequential TSMO	$470334.46_{\pm 9986.54}$	$1385.66_{\pm 36.50}$	$3259.70_{\pm 117.10}$		
3 cpus					
TSMO sync.	$470671.49_{\pm 10338.98}$	$1392.21_{\pm 38.62}$	$2828.90_{\pm 101.02}$	$26.44\% \leftrightarrow 39.13\%$	15.22%
TSMO async.	$470399.03_{\pm 11059.48}$	$1388.15_{\pm 36.56}$	$1632.93_{\pm 100.70}$	$27.21\% \leftrightarrow 37.96\%$	99.62%
TSMO coll.	$449481.89_{\pm 8385.78}$	$1310.20_{\pm 24.09}$	$3921.47_{\pm 147.67}$	$51.15\% \leftrightarrow 15.71\%$	−16.88%
6 processors					
TSMO sync.	$475855.56_{\pm 11541.97}$	$1392.70_{\pm 40.40}$	$2693.00_{\pm 115.12}$	$23.35\% \leftrightarrow 43.15\%$	21.04%
TSMO async.	$467515.13_{\pm 10063.36}$	$1379.51_{\pm 35.15}$	$1381.03_{\pm 66.44}$	$26.21\% \leftrightarrow 40.32\%$	136.03%
TSMO coll.	$435236.65_{\pm 6480.02}$	$1276.22_{\pm 20.18}$	$4245.23_{\pm 151.00}$	$59.48\% \leftrightarrow 8.79\%$	−23.21%
12 processors					
TSMO sync.	$477290.27_{\pm 11028.84}$	$1401.91_{\pm 39.34}$	$2617.40_{\pm 85.12}$	$20.71\% \leftrightarrow 46.34\%$	24.54%
TSMO async.	$469170.10_{\pm 10339.83}$	$1391.31_{\pm 37.23}$	$1894.67_{\pm 96.34}$	$23.66\% \leftrightarrow 43.49\%$	72.05%
TSMO coll.	$427434.85_{\pm 5902.75}$	$1253.45_{\pm 16.29}$	$4556.53_{\pm 170.84}$	$68.25\% \leftrightarrow 4.44\%$	−28.46%

TABLE III

Results on the 600 city extended Solomon problems with small time windows (C1, R1). Each problem was run 30 times, the maximum number of evaluations was set to 100,000, neighborhood size was set to 200 and if no better solution was found after 100 iterations, a restart with an individual from the memory was attempted. The size of the archive was set to 20 as was the value of the tabu tenure

Algorithm	distance	vehicles	runtime	coverage	speedup
Sequential TSMO	$365740.27_{\pm 10886.61}$	$575.34_{\pm 55.43}$	$4180.83_{\pm 123.29}$		
3 processors					
TSMO sync.	$367111.04_{\pm 9924.14}$	$606.01_{\pm 60.65}$	$3618.60_{\pm 102.23}$	$25.86\% \leftrightarrow 39.10\%$	15.53%
TSMO async.	$365969.76_{\pm 10437.97}$	$577.95_{\pm 55.42}$	$2419.13_{\pm 85.49}$	$26.66\% \leftrightarrow 38.16\%$	72.82%
TSMO coll.	$346291.10_{\pm 7559.73}$	$474.33_{\pm 32.44}$	$4796.93_{\pm 133.17}$	$49.33\% \leftrightarrow 15.39\%$	−12.84%
6 processors					
TSMO sync.	$368165.41_{\pm 10946.54}$	$599.71_{\pm 57.30}$	$3459.63_{\pm 99.68}$	$22.38\% \leftrightarrow 43.04\%$	20.85%
TSMO async.	$359929.42_{\pm 9615.51}$	$587.97_{\pm 53.87}$	$1970.40_{\pm 58.23}$	$25.36\% \leftrightarrow 41.09\%$	112.18%
TSMO coll.	$336810.21_{\pm 7710.00}$	$423.13_{\pm 20.77}$	$5113.73_{\pm 123.81}$	$59.68\% \leftrightarrow 7.55\%$	−18.24%
12 processors					
TSMO sync.	$369626.25_{\pm 11094.96}$	$632.69_{\pm 67.77}$	$3389.37_{\pm 96.31}$	$21.02\% \leftrightarrow 46.02\%$	23.35%
TSMO async.	$362516.18_{\pm 10185.95}$	$575.03_{\pm 57.08}$	$2195.37_{\pm 73.09}$	$24.21\% \leftrightarrow 42.50\%$	90.44%
TSMO coll.	$328492.92_{\pm 7369.84}$	$412.19_{\pm 19.67}$	$5361.77_{\pm 110.39}$	$64.46\% \leftrightarrow 4.21\%$	−22.03%

TABLE IV

Results on the 600 city extended Solomon problems with small time windows (C2, R2). Each problem was run 30 times, the maximum number of evaluations was set to 100,000, neighborhood size was set to 200 and if no better solution was found after 100 iterations, a restart with an individual from the memory was attempted. The size of the archive was set to 20 as was the value of the tabu tenure