A Parallel Multi-neighborhood Cooperative Tabu Search for Capacitated Vehicle Routing Problems

Jianyong Jin[1] Teodor Gabriel Crainic[2] Arne Løkketangen[1]

[1]Molde University College, Specialized University in Logistics, Norway
{Jianyong.Jin, Arne.Lokketangen}@himolde.no

[2]School of Management, UQAM & Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, Montréal, QC, Canada
TeodorGabriel.Crainic@cirrelt.ca

Abstract

This paper presents a parallel tabu search algorithm that utilizes several different neighborhood structures for solving the capacitated vehicle routing problem. Single neighborhood or neighborhood combinations are encapsulated in tabu search threads and they cooperate through a solution pool for the purpose of exploiting their joint power. The computational experiments on 32 large scale benchmark instances show that the proposed method is highly effective and competitive, providing new best solutions to four instances while the average deviation of all best solutions found from the collective best results reported in the literature is about 0.22%. We are also able to associate the beneficial use of special neighborhoods with some test instance characteristics and uncover some sources of the collective power of multi-neighborhood cooperation.

Keywords: Routing, Parallel metaheuristics, Multi-neighborhood, Cooperative search, Solution pool.

1 Introduction

The vehicle routing problem (**VRP**) describes the allocation of transportation tasks to a fleet of vehicles, and the simultaneous routing of each vehicle. The VRP was first described by Dantzig and Ramser (1959), and has been proved NP-hard by Lenstra and Kan (1981). Due to its high industrial applicability and complexity, the VRP has been the object of numerous studies and a great number of papers have proposed solution methods. These methods comprise both exact and heuristic algorithms. Since the VRP is NP-hard, it is not always possible to solve instances to optimality within limited computing time. Exact algorithms have been used to solve the VRP instances with up to about 100 customers (Laporte, 2007). For larger problems, heuristics and metaheuristics are

more appropriate, especially tabu search (**TS**)(Glover and Laguna, 1997), which has often been used successfully. For more information on the VRP, its solution methods and the recent work, we refer to the books of Toth and Vigo (2002), Golden et al. (2008) and the survey paper of Laporte (2007).

Among the solution methods for solving the VRP, one trend is to adopt parallel algorithms. For instance, parallel algorithms are increasingly applied in real time vehicle routing contexts where solutions have to be timely generated (Ghiani et al., 2003). Unlike sequential algorithms which run on a single processor and are executed sequentially, parallel algorithms run multiple processes simultaneously on available processors with the common goal of solving a given problem instance. Crainic (2008) describes the main strategies used on this group of algorithms and also provides an up-to-date survey of contributions to this rapidly evolving field. The author also points out that parallel algorithms can both speed up the search and improve the robustness and the quality of the solutions attained. Thus it would be advantageous to make use of parallelism.

Another feature of the latest metaheuristics for the VRP is to use multiple neighborhoods (e.g., Li et al., 2005; Kytöjoki et al., 2007; Mester and Bräysy, 2007; Groër et al., 2011). From the outcome of these algorithms, one may perceive that it is beneficial to employ multiple neighborhoods. Indeed, each neighborhood can be used to improve or modify a solution in its particular way such as reinserting a node, swapping two nodes and so forth. For a particular instance, at a certain stage, a specific neighborhood may be more effective than the others by leading the search through its own distinct search trajectory and producing better solutions. We term such a capability of a neighborhood as its effectiveness. Moreover, it is also noticeable that in these methods multiple neighborhoods are used in serial manner, in that each neighborhood is used one after another following a fixed or randomized sequence. One may wonder whether it can be more effective to use multiple neighborhoods in a parallel way instead. The objective of this paper is to explore the strategy of utilizing multiple neighborhoods in a parallel setting and analyze their effectiveness for solving the capacitated vehicle routing problem (**CVRP**).

The CVRP, as the classical version of the VRP, is defined on a graph $G = (N, A)$ where $N = \{0, \ldots, n\}$ is a vertex set and $A = \{(i, j) : i, j \in N\}$ is an arc set. Vertex 0 is the depot where the vehicles depart from and return to. The other vertices are the customers which have a certain demand d to be delivered (or picked up). The travel cost between customer i and j is defined by $c_{ij} > 0$. The vehicles are identical. Each vehicle has a capacity of Q. The objective is to design a least cost set of routes, all starting and ending at the depot. Each customer is visited exactly once. The total demand of all customers on a route must be within the vehicle capacity Q. Some CVRP instances may have an additional route duration limit constraint, restricting the duration (or length) of any route not to exceed a preset bound D. The method presented in this paper is able to solve problems both without and with route duration limit constraint.

The main contribution of this paper is the development of an effective parallel multi-neighborhood tabu search (**PMNTS**) method which contains several cooperative TS threads, each using a single neighborhood or a neighborhood combination. The experiments on 32 large scale CVRP benchmarks demonstrate that the proposed algorithm is effective and competitive. It finds new best solutions for four instances while the rest of the results are highly comparable to the best solutions reported in the literature. In addition, we are

also able to associate the beneficial use of special neighborhoods with some test instance characteristics and uncover some origins of the collective power of multi-neighborhood cooperation.

The remainder of this paper is organized as follows. In the next section our problem solving methodology is introduced. After that, Section 3 describes several variants developed for analyzing the performance of the proposed method. Then Section 4 presents the algorithm configurations and the computational results. Finally, conclusions are presented in Section 5.

2 Description of the parallel algorithm

In this section, the proposed parallel multi-neighborhood cooperative tabu search metaheuristic is introduced. We start with a general description of the algorithm, and then the description of the main search phase is provided in Section 2.1. After that, Section 2.2 introduces the cooperating TS threads. Then the initial solution phase is described in Section 2.3.

The general scheme of the proposed algorithm is displayed in Algorithm 1. The method consists of two phases. The goal of the first phase is to create a feasible starting solution (Line 1 of Algorithm 1). In the second phase (Line 2-8 of Algorithm 1), the starting solution is gradually improved by the joint effort of four different TS threads working in parallel (Line 4). Each TS thread utilizes a distinct neighborhood or neighborhood combination for inter-route moves and thus follows a different search strategy. These threads exchange the best solutions found periodically through a solution pool (Line 5-6). This phase is the core component and main search phase of the proposed method.

Algorithm 1: PMNTS

1: Create a feasible starting solution
2: Initialize the solution pool and the parallel tabu search threads
3: **while** termination conditions not met **do**
4: TS threads attempt to improve the starting solution independently
5: TS threads pause and export best solutions found to the solution pool
6: The solution pool selects new starting solution and send it to TS threads
7: **end while**
8: Return the best solution in the solution pool

2.1 The main search phase

In the main search phase, the four TS threads cooperate through a solution pool, focusing on improving the starting solution.

For a multi-thread metaheuristic, generally there are three types of parallelization strategy. The first type is independent multi-search in which multiple threads run simultaneously without any information exchange. The second type is allowing search threads to

exchange information directly between each other. The third strategy is to exchange information between threads through a solution pool. The reason we select the last strategy is twofold. First, it has been stated that the third strategy is more effective and parsimonious than the other two (Crainic, 2008). Second, some TS threads included in the proposed algorithm are not designed for independent use. Their individual performances are quite poor. Information exchange seems to be a necessity for them to work well in this context. The collaboration between the TS threads and the solution pool is depicted in Figure 1. The arrows in the figure represent the solution exchange between the TS threads and the solution pool.

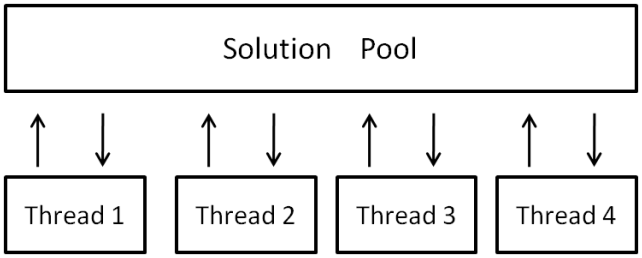


Figure 1: Collaboration diagram

The solution pool, which can be called a solution warehouse as well, keeps the best solutions sent by each search thread and sorts these solutions according to their quality, eliminating duplicate solutions and providing new starting solutions for each search thread. Each search thread only communicates with the solution pool.

To create moments for communications with each search thread, the second phase is partitioned into a certain number of segments (or search periods). During each segment these four search threads operate a certain amount of time, then they stop to export their best solutions found to the solution pool and attain a new solution from where to start the next segment. If there is a new best solution in the pool, this new best will be the starting solution for the next segment. Though a best solution may not always lead to another best one, we prefer to explore it first rather than a solution randomly selected. When there are no new best solutions in the pool, the new starting solution is selected by considering two criteria, namely the solution quality (the total travel distance) and the difference from the current best solution. The solution difference is measured by the number of different edges. Here, solutions with a certain amount (a parameter) of different edges are accepted so as to resume the search from sufficiently distinct solutions compared to where the search stopped. The solutions in the pool are sorted from the best to the worst according to the total travel distance, and each solution also has a flag that indicates whether it has been previously improved upon. This flag is used to avoid repeatedly selecting the same solution to resume the search. The solutions are checked from the best to the worst, and the first solution which has not been used before and satisfies the difference requirement will be selected for the next segment. The rationale behind starting from the same solution in all threads are threefold. First, each search thread can start from the best solution found so that all threads can reach a promising part of the search space quickly. Second, the search threads are kept in the same area of the search space to intensively search that area. Third, for a given starting solution, the neighborhood that is the most appropriate for generating improvement will have the opportunity to work on the solution.

In terms of the taxonomy introduced by Crainic and Nourredine (2005) for parallel metaheuristics, our algorithm fits into the *pC/KS/SPDS* classification. The first dimension *pC* indicates the global search is controlled by multiple cooperative threads. The second dimension *KS* stands for knowledge synchronization and refers to the fact that multiple threads share information synchronously. The last dimension *SPDS* indicates multiple search threads make use of different search strategies from the same initial solution during each period.

2.2 The tabu search threads

In this sub-section, the common features and the differences between the TS threads included in the proposed algorithm are provided.

2.2.1 The common features of the tabu search threads

The TS threads included in the proposed method are developed on the basis of the granular tabu search that was first introduced by Toth and Vigo (2003). Granular TS is a mechanism which is able to reduce the computational effort, especially for large instances by not considering some of the unpromising solution components (in their case, the long edges). In this paper, a similar, but somewhat different granular neighborhood is implemented. It is to select a set of the nearest neighbors (plus the depot) for each customer, and then only moves involving one member of this nearest neighbors set will be considered. The size of the nearest neighbors set (denoted as SNNS) can be selected by considering the instance characteristics and the requirements of the solution quality (or the time available for computation) as suggested by Branchini et al. (2009). The other aspects of TS framework are described below.

Inter-route neighborhood structure
Three basic neighborhood structures for inter-route operations, which are commonly used in the previously published metaheuristics for the VRP, are selected and implemented in the TS threads. The basic idea of each neighborhood structure is described as follows.

- Reinsertion (Savelsbergh, 1992) refers to relocating a customer node from one route to another route.
- Exchange (Osman, 1993) swaps two customer nodes from two routes.
- 2-opt* (Potvin and Rousseau, 1995) combines two routes so that the last customers of a given route are introduced after the first customers of another route, i.e. exchanging the head or tail part of two routes.

Each TS thread included in the proposed algorithm utilizes one or a distinct subset of these neighborhoods. Thus each thread follows a different search strategy. When a TS thread incorporates more than one neighborhoods for inter-route moves, one of the included neighborhoods is randomly selected according to a certain probability at each iteration.

Neighborhood generation and evaluation
For each inter-route neighborhood structure, at each iteration, all customer nodes are used to generate neighboring solutions. For each customer node u, assuming in route $R1$ in the current solution s, find the nodes which are in its nearest neighbors set but not in route $R1$. The set of these nodes is denoted as Ω. For each node v in Ω in route $R2$,

- Reinsertion: move u to route $R2$ after node v.
- Exchange: swap the positions of u and v.
- 2-opt*: denote the node after u in route $R1$ as u', denote the node before v in route $R2$ as v', replace edge (u, u') and (v, v') with edge (u, v) and (u', v').

To explore the solution space more thoroughly, infeasible intermediate solutions are allowed. For this purpose, capacity and route length constraints are relaxed and their violations are penalized in the objective function. This augmented objective function is computed as $F(s) = C(s) + \alpha Q(s) + \beta D(s)$, where $C(s)$ is the total travel distance, $Q(s)$ and $D(s)$ denote the total violations of capacity and route length constraints respectively, α and β are penalty multipliers. The values of the penalty multipliers are self-adjusted during the course of the search as described by Toth and Vigo (2003). The neighboring solutions (either feasible or infeasible) generated by each neighborhood operator are evaluated in terms of the augmented objective function.

In addition, to speed up the search, the neighborhood exploration and evaluation for inter-route moves are parallelized by decomposing the computational tasks to different threads.

Solution acceptance and tabu mechanism
Among the neighboring solutions, the best move in terms of the augmented objective function that is non-tabu or satisfies the aspiration criterion is accepted. The aspiration criterion overrides the tabu status of a move if this move leads to a new best solution.

The tabu list is neighborhood dependent. The tabu list tenure tt of each neighborhood is set to be proportional to the number of customer nodes in the instance. For reinsertion, if u is relocated, u is declared tabu for tt iterations and any moves relocating u cannot be performed unless it satisfies the aspiration criterion. For exchange move swapping u and v, the two nodes are declared tabu and any moves swapping u and v cannot be performed unless it satisfies the aspiration criterion. For 2-opt* moves adding edge (u, v) and (u', v'), node u, v and v' are declared tabu, any moves adding edge (u, u') and (v, v') are forbidden unless it satisfies the aspiration criterion.

Route refinement
In the TS threads, at each iteration, after an inter-route move, the two modified routes are refined separately by an intra-route optimization procedure. The procedure consists of two simple heuristics developed by implementing 2-opt (Flood, 1956) and reinsertion (Savelsbergh, 1992) neighborhood structures in local search setting. The two heuristics are applied to a route alternately. The heuristic using 2-opt repeatedly eliminates two edges and adds two new edges, improving moves are accepted until no improvement can be found. The heuristic using reinsertion seeks improvement by relocating a node to

another position, if a move reduces the route length, then it is accepted. The procedure terminates when no improvement can be found.

Solution exchange
The TS threads pause and exchange solutions with the solution pool after running for a certain amount of time. Each TS thread exports its best solution and receives a new solution to resume the search from.

Search process
Each tabu search thread starts off with the starting solution s_i created in the first phase. At each iteration, a neighborhood is selected to generate a set of neighbors and the least cost non-tabu solution $\bar{s}$ is selected to replace the current solution s. Then the reverse moves are declared tabu and the routes just modified are refined by the intra-route optimization procedure. In addition, if the current solution s is feasible and better than the best solution s^*, replace s^* with s. Moreover, periodically each tabu search thread stops to exchange solutions with the solution pool and uses the received solution to replace its current solution and the best solution, the tabu lists are re-initialized. This process is summarized in Algorithm 2.

Algorithm 2: Tabu search

1: Set $s^* = s_i$, $s = s_i$, $C(s^*) = C(s_i)$
2: Initialize tabu lists and penalty multipliers
3: **while** termination conditions not met **do**
4: Select a neighborhood and SNNS for inter-route moves
5: Generate and evaluate neighboring solutions
6: Select a neighboring solution $\bar{s}$ that minimize $F(\bar{s})$ and is non-tabu or satisfies the aspiration criterion, set $s = \bar{s}$
7: Set the reverse moves tabu for tt iterations
8: Refine the routes modified
9: If s is feasible and $C(s) < C(s^*)$, set $s^* = s$; $C(s^*) = C(s)$
10: Update penalty multipliers
11: If reaching the time limit, pause and exchange solutions with the solution pool, reset s^*, s, $C(s^*)$ and tabu lists
12: **end while**

2.2.2 The differences between the tabu search threads

As summarized in Table 1, each TS thread utilizes a distinct neighborhood or neighborhood combination for inter-route moves and thus each thread plays a different role in the cooperative search.

Thread 1 utilizes two neighborhoods (reinsertion and 2-opt*) for inter-route moves in serial fashion. In each iteration a neighborhood is randomly selected according to a certain probability. The probability of using reinsertion neighborhood is much higher than using 2-opt*. Preliminary computational experiments show that using such a combination is more effective than using only reinsertion neighborhood, especially for the instances with

Table 1: The features of each TS thread

TS thread	Neighborhood used for inter-route	Role
Thread 1	Reinsertion, 2-opt*.	Main improving thread.
Thread 2	2-opt*.	Assistant improving thread.
Thread 3	Exchange	Assistant improving thread.
Thread 4	Solution perturbation + Reinsertion, 2-opt*, Exchange	Diversifying the search.

tight constraints. For the actual probability values used, see Section 4.3. One advantage of such serial neighborhood cooperation can be that this mechanism allows different neighborhoods to work one after another on intermediate infeasible solutions which may lead to good feasible solutions.

Thread 2 and Thread 3 are similar. They utilize one single neighborhood for inter-route moves trying to improve the given solution. These two neighborhoods can be quite effective for some instances at various search stages. Thus, with these two threads, the proposed method is able to identify good solutions for a broad variety of instances.

Thread 4 is different from the other three. Its main task is to diversify the search process. This thread consists of an extra solution perturbation procedure. Before the normal TS procedure starts, the starting solution is perturbed first. The perturbation procedure consists of two steps. Firstly, a certain percentage of nodes of a solution are removed from the solution. The term *perturbation strength* is used to refer to this percentage. Half of those nodes are selected from the nodes which are connected to their distant neighbors while the other half are selected randomly. During the second step, reinsert the nodes removed one by one to a different route, next to one of its nearest neighbors so as to cause the least deterioration in terms of the augmented objective function value. The thread utilizes three neighborhoods (reinsertion, 2-opt* and exchange) for inter-route moves in serial fashion. The neighborhood selection rule is similar as in Thread 1.

2.3 The initial solution phase

As displayed in Algorithm 1, the goal of the first phase is to create a feasible starting solution. There are three steps in the phase. Step one creates an initial solution in which each customer is served individually by a separate route, i.e., the number of routes equals the number of customers. In step two, the routes are merged by a route reduction procedure so that the number of routes is reduced to a preselected level. In the route reduction procedure, there are two sub-steps. The first sub-step is to select the route with the smallest load, denote the route as $R1$. The second sub-step is to reinsert each customer node in $R1$ to another route. To this end, for each customer node in $R1$, use reinsertion neighborhood operator to generate and evaluate possible moves as described in Section 2.2.1, and then the best move in terms of the augmented objective function value is executed. Repeat the two sub-steps until the number of routes reaches the preselected level. The solutions generated in step two are often infeasible regarding either the capacity constraint or the route

length constraint. Thus, in step three, an attempt is made to restore the feasibility by a repair procedure. The repair procedure is a variant of the tabu search procedure described in Section 2.2.1, in which three neighborhoods (reinsertion, exchange and 2-opt*) are used. To focus restoring the feasibility, a special move evaluation and acceptance criterion is used for reinsertion and exchange neighborhoods. For the two neighborhoods, the neighbors are generated as described in Section 2.2.1, but they are evaluated in terms of three separate metrics, i.e., the total travel distance, the capacity constraint violations and the route length constraint violations. Among the available moves, only moves which result in the positive decrease in the total travel distance will be considered. If such moves exist, then the repair procedure checks whether there exist route length constraint violations. If there are route length constraint violations, the move leading to the neighboring solution with the lowest route length constraint violations is executed, otherwise, the move leading to the neighboring solution with the lowest capacity constraint violations is executed. The rationale behind this decision is that the capacity is relatively easier to restore. If such improving moves are not found, no moves will be executed. At each iteration, reinsertion neighborhood is applied first to search for improving moves. If no such moves can be found, then exchange neighborhood is applied. When neither reinsertion nor exchange neighborhood can find improving moves, 2-opt* neighborhood is applied. For this neighborhood operator, the moves are evaluated in terms of the augmented objective function value and the best non-tabu move will be accepted even though it may increase the total constraint violations. The first phase terminates when solution feasibility is attained.

In the proposed algorithm, the number of routes in the solutions is fixed during the subsequent search course after the route reduction procedure. The rationale behind this decision is twofold. First, the number of vehicles is often known beforehand in some VRP contexts. Second, to fix the number of tours facilitates the method to focus on minimizing the total travel distance. For situations where the number of vehicles is unknown, the proposed algorithm is also able to find solutions with the minimum number of routes by enclosing it in a binary search in the number of routes. For the CVRP benchmarks tested during computational experiments, the minimum route number of each problem reported in the literature is adopted for the sake of simplicity.

3 Several other variants

To evaluate the role of cooperation and to compare the effectiveness of using multiple neighborhoods in serial and parallel manner, we have also developed six other variants, i.e. four sequential variants (denoted as SV1, SV2, SV3 and SV4), two parallel variants (denoted as PV1 and PV2).

The four sequential variants are developed by removing the solution pool and 3 threads from the second phase of PMNTS. For instance, SV1 is developed by removing the solution pool, Thread 2, Thread 3 and Thread 4, thus only keeping Thread 1 at the second phase. SV2, SV3 and SV4 are developed analogously by only keeping Thread 2, Thread 3 and Thread 4 at the second phase respectively. Thus, in particular, SV4 corresponds to a sequential TS using all the neighborhoods used by the parallel version. These sequential variants have only one thread, run on a single processor and correspond to each of the four TS threads included in PMNTS respectively. The objective of generating these variants is

to evaluate the performance of each simple TS thread included in the proposed algorithm.

The first parallel variant PV1 is developed by removing Thread 4 and 2-opt* neighborhood at Thread 1 from PMNTS. It consists of only three threads in the second phase and each search thread utilizes a single neighborhood (reinsertion, exchange or 2-opt*) for inter-route moves respectively. The three TS threads cooperate through the solution pool. The solution pool functions exactly the same way as discussed above for PMNTS. The goal of generating this variant is to evaluate the effectiveness of using only one neighborhood for inter-route moves in each TS thread and explore the association between the beneficial use of special neighborhoods with test instance characteristics.

The second parallel variant PV2 is developed by replacing the original Thread 2 and Thread 3 at the second phase of PMNTS with clones of Thread 1. In the resulting parallel variant, three copies of Thread 1 cooperate with Thread 4 through the solution pool. The objective of developing this variant is to evaluate the performance of PMNTS without Thread 2 and Thread 3 in which single neighborhoods are used in a parallel manner.

These variants are tested with a subset of CVRP benchmark instances to compare their effectiveness. The results are discussed in Section 4.

4 Computational results

In this section we describe the experimental platform, the test data sets, the algorithm configurations, the experimental results and give some observations.

4.1 Experimental platform and implementation issues

The proposed algorithm is implemented in C++ and Intel Threading Building Blocks (**TBB**) libraries are used for parallelization. TBB is a C++ template library developed by Intel Corporation for writing software programs that take advantage of multi-core processors. More information about TBB can be found at http://www.threadingbuildingblocks.org/. The computational experiments are carried out on a computer with 2 Intel® Xeon® E5450 3.00GHZ CPUs(quad-core) and 8 GB of RAM. A master thread is used to control the global search process, launch the four search threads in phase 2 and manage the solution pool. Each search thread is run on one core. The rest of the available cores are used by the parallel neighborhood generation and evaluation procedures.

4.2 The test data sets

The computational tests were carried out using the CVRP benchmarks of Golden et al. (1998) and Li et al. (2005). The 20 benchmark instances of Golden et al. (1998) have 200 to 483 customers. The first eight instances also have route length restrictions. Each instance is based on a simple geometric structure: eight instances have customers located in concentric circles around the depot, four instances have customers located in concentric squares with the depot located in one corner, four instances have customers located in concentric squares around the depot, and four instances have customers located in a six-pointed star around the depot. The benchmark instances of Li et al. (2005) have 560 to 1200 customers and route length restrictions, and their geometric structure is based on

concentric circles around the depot. For each instance, the algorithm was executed 10 times with different random seeds and both the best and average results are reported.

4.3 The algorithm configurations

The configurations of PMNTS and the other variants are introduced below.

4.3.1 The configurations of PMNTS

The parameters of the proposed parallel multi-neighborhood cooperative tabu search algorithm are shown in Table 2.

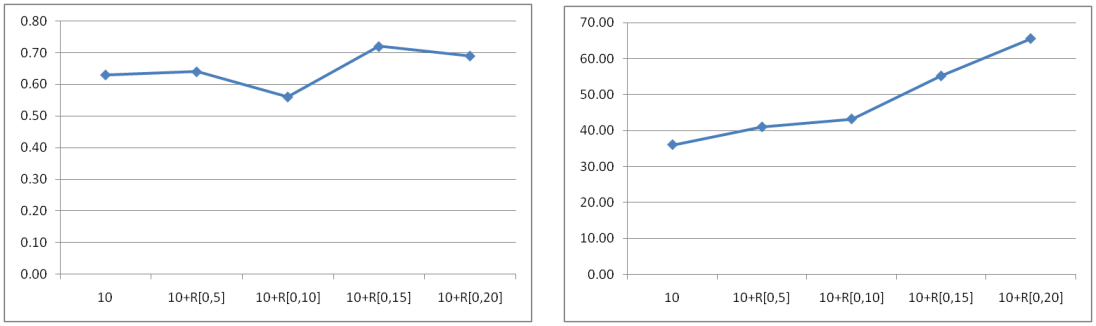
Table 2: The parameters of the proposed algorithm

Parameters	**First phase**	**Second Phase**			
		Thread1	**Thread2**	**Thread3**	**Thread4**
Tabu tenure	R: 0.07N E: 0.07N O: 0.135N	R: 0.05N O: 0.1N	O: 0.135N	E: 0.07N	R: 0.07N E: 0.07N O: 0.1N
Neighborhood probability		R: $^6/_7$ O: $^1/_7$	O: 1	E: 1	R: $^5/_7$ E: $^1/_7$ O: $^1/_7$
Nearest neighbors set size	24	10 + random [0, 10]			
Perturbation strength		Thread4 : 0.2			
Solution difference		at least 10% different edges			
Time for a segment		the time Thread 1 runs $250\sqrt{N}$ iterations			
Segment number		60			

R represents reinsertion, E stands for exchange and O represents 2-opt*.

Here, N represents the number of customer nodes in the instance. Tabu tenure is set to be proportional to N. For example, in Thread 1, tabu tenure of reinsertion neighborhood is 0.05N while 2-opt* neighborhood uses 0.1N. In Thread 1, the probability of selecting reinsertion neighborhood is $^6/_7$ while the probability of selecting 2-opt* neighborhood is $^1/_7$. The size of the nearest neighbors set is calculated as 10 plus a uniform random number from the interval [0, 10] at each iteration in the second phase. In Thread 4, 20% of nodes are relocated in the perturbation procedure. As for solution difference, the requirement is that a solution has at least 10% percent different edges from the current best solution. In each segment, the running time is controlled by Thread 1. After running $250\sqrt{N}$ iterations, the thread stops, and the other three threads are also terminated. The whole search terminates after 60 segments or when there is no improvement for 15 consecutive segments. The values of these parameters are tuned through extensive testing on the problems 1,4,5,8,9,12,13,16,17 and 20 of Golden et al. (1998) based on preliminary computational experiments. These instances are selected since they differ in both instance size and customer location structure. As an example, Figure 2 shows the impact of the size

of nearest neighbors set on the solution quality and time. The solution quality is measured with the average deviation of all instances tested from the best known solutions while the time is the average running time of all instances tested in minutes. It is noticeable that larger values cause longer running times but not necessarily better solution quality. The results also show that the solution quality is not so sensitive to the parameter. Among the values tested, 10+random [0,10] is chosen. The other parameters are calibrated in a similar way.



(a) Quality (b) Time

Figure 2: Impact of the size of nearest neighbors set

4.3.2 The configurations of the variants

For the four sequential variants, the parameters of the first phase remain identical as in PMNTS. At the second phase, the parameters are set to the same as for the corresponding TS thread, i.e., the parameters of the second phase of SV1, SV2, SV3 and SV4 are set to the same as for Thread 1, Thread 2, Thread 3 and Thread 4 in PMNTS respectively. However, each sequential variant is set to be executed for $4 \times 60 \times 250\sqrt{N}$ iterations so that it utilizes approximately the same amount of computing resource as PMNTS.

As for the first parallel variant PV1, the parameter setting is the same with PMNTS except that there is no Thread 4 and only reinsertion neighborhood in Thread 1 (the probability of using reinsertion is 1).

With regard to the second parallel variant PV2, the configurations are the same as PMNTS except that Thread 2 and Thread 3 in PV2 are clones of Thread 1. The parameters for these three threads are set to the same as for Thread 1 in PMNTS.

4.4 Results for the benchmarks of Golden et al. (1998)

In Table 3 we compare the results for the 20 benchmark instances of Golden et al. (1998) against previously published work. In this table, the first column describes the problem instance (problem number and number of nodes). The second column lists the best known solutions previously reported in the literature. The third to ninth columns provide the best results reported in each paper. The remaining columns give the average results, average time and best results of PMNTS. The third last row presents the average deviation of all instances from the best known solutions. The second last row provides running time measures. For the methods providing the best results of multiple runs (Li et al., 2005;

Table 3: Comparison of results for the benchmarks of Golden et al. (1998)

Problem	Previous best known	Li et al. (2005)	Psinger and Ropke (2007)	Mester and Bräysy (2007)	Kytöjoki et al. (2007)	Nagata and Bräysy (2009)	Groër et al. (2011) 4pc	Groër et al. (2011) 129pc	PMNTS aver.	PMNTS aver. time (min)	PMNTS best
1(240)	5623.47	5666.42	5650.91	5627.54	5867.84	5626.81	5644.44	5623.47	5627.54	16.54	5623.47
2(320)	8431.66	8469.32	8469.32	8447.92	8476.26	8431.66	8447.92	8435.00	8447.15	29.72	**8419.50**
3(400)	11036.22	11145.80	11047.01	11036.22	11043.41	11036.22	11036.22	11036.22	11080.60	58.44	**11030.80**
4(480)	13592.88	13758.08	13635.31	13624.52	13631.72	13592.88	13624.52	13624.52	13666.84	87.03	13615.20
5(200)	6460.98	6478.09	6466.68	6460.98	6460.98	6460.98	6460.98	6460.98	6464.40	10.47	6460.98
6(280)	8404.26	8539.61	8416.13	8412.88	8415.67	8404.26	8412.90	8412.90	8468.10	23.96	**8403.25**
7(360)	10156.58	10289.72	10181.75	10195.56	10297.66	10156.58	10195.59	10195.59	10209.24	62.16	10184.40
8(440)	11643.90	11920.52	11713.62	11663.55	11872.64	11691.06	11680.31	11649.89	11725.82	86.68	11671.00
9(255)	579.71	588.25	585.14	583.39	620.67	580.42	583.37	579.71	583.15	18.70	581.73
10(323)	737.28	749.49	748.89	741.56	784.77	738.49	742.43	737.28	739.97	38.65	738.50
11(399)	913.35	925.91	922.70	918.45	986.80	914.72	917.91	913.35	917.50	58.96	914.98
12(483)	1102.76	1128.03	1119.06	1107.19	1209.02	1106.76	1117.05	1102.76	1112.44	72.84	1109.93
13(252)	857.19	865.20	864.68	859.11	925.81	857.19	858.89	857.19	864.45	18.82	861.92
14(320)	1080.55	1097.78	1095.40	1081.31	1155.19	1080.55	1081.24	1080.55	1084.59	27.94	1082.52
15(396)	1338.00	1361.41	1359.94	1345.23	1461.49	1342.53	1346.45	1338.00	1353.07	38.07	1351.13
16(480)	1613.66	1635.58	1639.11	1622.69	1742.86	1620.85	1624.42	1613.66	1632.88	55.78	1629.78
17(240)	707.76	711.74	708.90	707.79	726.01	707.76	707.79	707.76	708.46	15.83	707.83
18(300)	995.13	1010.32	1002.42	998.73	1077.53	995.13	998.66	995.13	1002.53	30.81	1000.27
19(360)	1365.60	1382.59	1374.24	1366.86	1444.51	1365.97	1369.34	1365.60	1368.22	36.39	1367.31
20(420)	1818.25	1850.92	1830.80	1820.09	1938.12	1820.02	1824.98	1818.25	1830.10	49.62	1827.39
Aver. deviation %		1.33	0.47	0.26	4.76	0.11	0.36	0.04	0.53		0.28
Time min		3.39	108.00	24.40	0.02	355.90	25.00	25.00		41.87	418.70
Runs per instances		3	10	1	1	10	5	5			10

Pisinger and Ropke, 2007; Nagata and Bräysy, 2009; Groër et al., 2011) and the best results of PMNTS, the table displays the total time of multiple runs, while the average time for a single run is provided for Mester and Bräysy (2007), Kytöjoki et al. (2007) and the average results of PMNTS. The last row shows the number of runs performed for each instance in each algorithm.

From the table, we see PMNTS found new best solutions to 3 problems (numbers in bold font) while the average deviation of the best solutions of PMNTS from the previous collective best known solutions is 0.28%. In terms of this metric, our results are better than Li et al. (2005), Pisinger and Ropke (2007), Kytöjoki et al. (2007) and Groër et al. (2011) using 4 processors, but slightly worse than Mester and Bräysy (2007), Nagata and Bräysy (2009) and Groër et al. (2011) using 129 processors.

4.5 Results for the benchmarks of Li et al. (2005)

The results for the 12 benchmark instances of Li et al. (2005) are presented in Table 4. The format of this table is the same as for Table 3. As in Table 3, the best results reported in six recently published papers are also displayed in Table 4 for comparison. In particular, the total time of multiple runs is provided in the table for the method proposed in Dorronsoro et al. (2007) similar to the other methods also providing the best results of multiple runs.

For this set of instances, the proposed algorithm found a new best solution to one problem (numbers in bold font). The average deviation of our best solutions found from the previous best known is 0.12%. According to this measurement, our results are better than most previous works except Groër et al. (2011) using 129 processors.

Table 4: Comparison of results for the benchmarks of Li et al. (2005)

Problem	Previous best known	Li et al. (2005)	Psinger and Ropke (2007)	Mester and Bräysy (2007)	Kytöjoki et al. (2007)	Dorronsoro et al. (2007)	Groër et al. (2011) 4pc	Groër et al. (2011) 129pc	PMNTS aver.	PMNTS aver. time (min)	PMNTS best
21(560)	16212.74	16602.99	16224.81	16212.74	16221.22	16212.83	16212.83	16212.83	16247.82	98.63	16220.00
22(600)	14584.42	14651.27	14631.08	14597.18	14654.87	14652.28	14631.73	14584.42	14618.83	121.69	14598.70
23(640)	18801.12	18838.62	18837.49	18801.12	18810.72	18801.13	18801.13	18801.13	18883.80	139.82	18829.80
24(720)	21389.33	21616.25	21522.48	21389.33	21401.41	21389.43	21390.63	21389.43	21427.93	88.08	21399.00
25(760)	16763.72	17146.41	16902.16	17095.27	17358.18	17340.41	17089.62	16763.72	16826.62	176.35	16781.70
26(800)	23971.74	24009.74	24014.09	23971.74	23996.86	23977.73	23977.73	23977.73	24127.10	103.54	23986.10
27(840)	17433.69	17823.40	17613.22	17488.74	18233.93	18326.92	17589.05	17433.69	17522.93	101.37	**17432.30**
28(880)	26565.92	26606.11	26791.72	26565.92	26592.05	26566.04	26567.23	26566.03	26609.50	136.17	26574.40
29(960)	29154.34	29181.21	29405.60	29160.33	29166.32	29154.34	29155.54	29154.34	29190.08	188.64	29162.70
30(1040)	31742.51	31976.73	31968.33	31742.51	31805.28	31743.84	31743.84	31742.64	31772.95	252.06	31753.40
31(1120)	34330.84	35369.17	34770.34	34330.84	34352.48	34330.94	34333.37	34330.94	34384.17	246.23	34340.50
32(1200)	36919.24	37421.44	37377.35	36928.70	37025.37	37423.94	37285.90	37185.85	37305.33	272.59	37204.80
Aver. deviation %		1.18	0.68	0.20	0.80	0.87	0.35	0.06	0.35		0.12
Time min		9.60	498.00	104.30	0.10	3660.00	25.00	25.00		160.43	1604.30
Runs per instances		3	10	1	1	2	5	5			10

4.6 Algorithm performance analysis

In Table 5, the average results of the six variants for a subset of test instances are compared together with PMNTS. First, by comparing the outcome of the four sequential variants with PMNTS, it is evident that PMNTS outperforms its component TS threads both in the solution quality and wall-clock time. In particular, although SV4 uses all neighborhood structures implemented in PMNTS, its performance appears worse than PMNTS. In addition, when comparing the performance of SV1, SV4 with PV1, we can see that PV1 has identified better solutions to four instances (problem 9, 12, 13 and 16) than the two sequential variants even though the outcome of PV1 overall seems to be inferior. This observation indicates that it is advantageous to utilize multiple neighborhoods both in serial manner (in SV1 and SV4) and in parallel manner (in PV1). Moreover, although SV1(Thread 1) individually outperforms both SV2(Thread 2) and SV3(Thread 3), the performance of PV2, in which Thread 2 and Thread 3 are clones of Thread 1, turns out worse than PMNTS. This fact also demonstrates that it is beneficial for PMNTS to include Thread 2 and 3 in which 2-opt* and exchange neighborhood are used in parallel manner.

4.7 Observations on neighborhood effectiveness

From the computational experiments, a few patterns regarding neighborhood effectiveness have been observed. In this subsection, these observations are presented.

4.7.1 An example where 2-opt* neighborhood is effective

In Table 6, an example is provided in which the steps that the single-neighborhood parallel variant PV1 took to identify a high quality solution for instance Golden_benchmark_5 are listed. In the table, the values of the initial solutions of each step are given in the column marked *IniSolObj*. The values of the best solutions each thread finds at each step are given in the columns marked with the thread name. The second last column gives the value of the current overall best solution found so far while the improvement of each step is given

Table 5: Comparison of results of the six variants with PMNTS

Problem	Previous best known	SV1	SV2	SV3	SV4	PV1	PV2	PMNTS
1(240)	5623.47	5631.25	5695.42	5724.42	5632.70	5667.90	5627.05	5627.54
4(480)	13592.88	13784.47	13701.02	14425.23	13778.33	13894.88	13686.55	13666.84
5(200)	6460.98	6465.73	6468.92	6677.05	6462.83	6473.53	6466.68	6464.40
8(440)	11643.90	11730.05	11759.83	11933.42	11728.33	11774.16	11744.30	11725.82
9(255)	579.71	587.26	600.55	650.05	585.80	585.27	584.45	583.15
12(483)	1102.76	1121.88	1159.71	1269.29	1118.68	1114.21	1117.28	1112.44
13(252)	857.19	874.44	888.62	932.00	871.22	869.95	869.71	864.45
16(480)	1613.66	1650.05	1696.27	1803.57	1642.33	1638.92	1638.97	1632.88
17(240)	707.76	709.26	718.09	731.25	708.87	714.68	708.36	708.46
20(420)	1818.25	1837.88	1886.57	1886.32	1833.94	1871.28	1834.69	1830.10
Aver. deviation %		1.10	2.60	6.85	0.92	1.33	0.79	0.56
Aver. time min		189.31	337.61	330.16	213.04	31.23	42.63	43.20

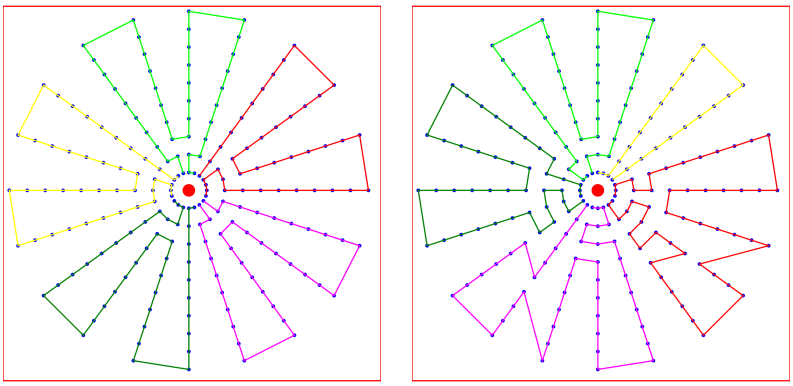
in the last column, which is the difference between the current overall best solutions of two steps. The numbers in bold indicate the best solutions among the three threads at each step. The numbers underlined represent the solutions that are not the best ones in terms of the value at each step but lead to a good solution later.

Table 6: An example search path for Golden_benchmark_5

Step	**IniSolObj**	**Thread1**	**Thread2**	**Thread3**	**BestObj**	**Improve**
1	7239.91	6934.63	**6860.00**	7126.31	6860.00	379,91
2	6860.00	6854.39	**6606.15**	6860.00	6606.15	253.85
3	6606.15	6590.60	**6547.60**	6601.92	6547.60	58.55
4	6547.60	6508.26	**6496.77**	6536.05	6496.77	50.83
5	6496.77	**6491.16**	6496.77	6496.77	6491.16	5.61
6	6536.05	**6508.26**	6516.23	6532.27	6491.16	0.00
7	6516.23	**6483.79**	6508.35	6492.21	6483.79	7.37
8	6508.35	**6472.38**	6475.19	6484.28	6472.38	11.41
9	6484.28	**6466.68**	6466.68	6484.28	6466.68	5.7
10	6532.27	6508.26	**6498.79**	6507.20	6466.68	0.00
11	6498.79	**6483.79**	6486.60	6489.40	6466.68	0.00
12	6489.40	**6483.79**	6486.59	6489.40	6466.68	0.00
13	6486.59	**6483.79**	6486.59	6484.89	6466.68	0.00
14	6484.99	**6460.98**	6479.38	6484.99	6460.98	5.8
Improvement		35.79 (4.6%)	743.14 (95.4%)	0 .00		778.93

From Table 6, we can see that Thread 2 that uses 2-opt* neighborhood has played an important role in identifying the best solution. In the first four steps, it always finds the best solutions among the three threads while Thread 1 using reinsertion neighborhood often provides the best during the late steps. Thread 2 has contributed 95.4% of the

total improvement directly. The underlying reason for this fact can be attributed to the characteristics of this instance. From its data, we can notice this instance has loose route length and capacity constraints and the location of its customers has a special geometric structure. For this instance, high quality solutions should have routes with balanced load and route length. However, due to its loose constraints, it is vulnerable to form unbalanced routes that result in low quality solutions. Since it is more effective in changing the structures of the routes, 2-opt* neighborhood can improve the solutions to this instance significantly. Figure 3 visualizes the features of two different solutions to this instance.



(a) A high quality solution (b) A low quality solution

Figure 3: Solution features of Golden_benchmark_5

4.7.2 An example where exchange neighborhood is effective

From the computational experiments, some cases show that exchange neighborhood can also find better solutions than other neighborhoods. A search path of PV1 for the instance Golden_benchmark_9 is instantiated in Table 7 to demonstrate such cases.

From Table 7, we can see exchange neighborhood has found the best solutions among the three threads at several steps. It makes the second largest contribution to the total improvement (4.7%). An early stage solution to this instance is showed in Figure 4, from which it is noticeable that the solution has routes overlapping each other. Such a feature facilitates exchange neighborhood to find good moves.

4.7.3 General trend of neighborhood effectiveness

To find the general trend of neighborhood effectiveness, the search steps that PV1 took to identify the best solution for ten instances are summarized in Table 8. In the table, the term *Solution path* refers to the sequence of neighborhoods used to find the best solution step by step from the beginning of the second phase. The header *Frequency of neighborhood used* represents how many times a neighborhood is used in a solution path.

From the two examples and Table 8, one may notice the following trends.

- The reinsertion neighborhood, given a certain amount of time, is often able to find better solutions than exchange and 2-opt* for most of the instances tested. Its performance is less related to the instance attributes.

Table 7: An example search path for Golden_benchmark_9

Step	IniSolObj	Thread1	Thread2	Thread3	BestObj	Improve
1	717.08	**594.08**	618.94	688.38	594.08	123.00
2	594.08	591.89	594.08	**590.84**	590.84	3.24
3	590.84	**589.46**	589.66	589.97	589.46	1.38
4	589.46	588.94	589.46	**588.48**	588.48	0.98
5	588.48	**588.07**	588.23	588.48	588.07	0.41
6	588.07	**586.90**	588.07	588.07	586.90	1.17
7	589.66	589.48	589.66	**589.29**	586.90	0.00
8	589.29	589.20	589.29	**589.19**	586.90	0.00
9	589.19	**587.02**	589.19	589.19	586.90	0.00
10	587.02	585.96	587.02	**585.79**	585.79	1.11
11	589.97	**589.15**	589.29	589.87	585.79	0.00
12	589.15	588.53	**587.91**	588.85	585.79	0.00
13	588.53	587.71	**587.36**	588.53	585.79	0.00
14	587.71	587.71	**585.36**	586.78	585.36	0.43
15	585.36	585.36	585.36	**584.44**	584.44	0.92
Improvement		125.96 (95.0%)	0.43 (0.3%)	6.25 (4.7%)		132.64

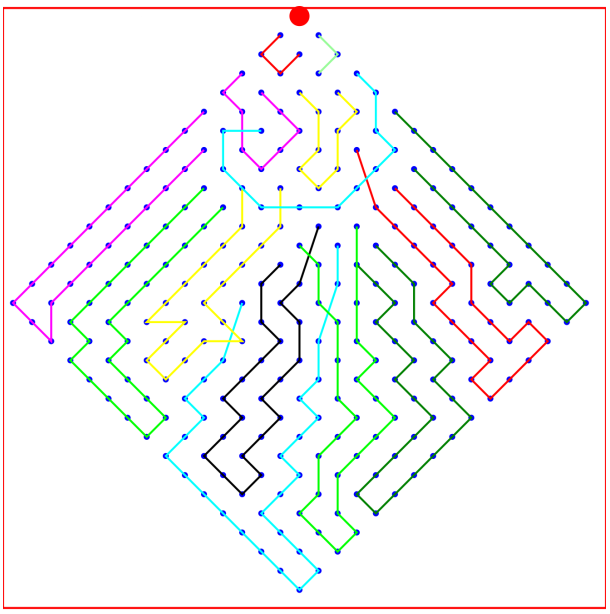


Figure 4: Solution features of Golden_benchmark_9

Table 8: The search paths of ten instances

Instance	Solution value	Solution path	Frequency of neighborhood used		
			Reinsertion	2-opt*	Exchange
Golden1	5646.89	R/O/R/R/R/E	4	1	1
Golden4	13839.10	R/R/O/E/E/O/R/R	4	2	2
Golden5	6460.98	O/O/O/E/E/O/E/O /E/R	1	5	4
Golden8	11768.4	R/E/O/R/R/R/R	5	1	1
Golden9	584.44	R/E/E/R/R/R/O/E	4	1	3
Golden12	1116.48	R/R/R/E/R/E/R/E /R/R/E/O/E	7	1	5
Golden13	868.23	O/R/O/R/R/O/R/E /E/R/E	5	3	3
Golden16	1629.38	R/E/E/O/R/O/R/E /R/O/E/E/O/R/R/E	6	4	6
Golden17	708.81	R/R/R/E/R/R/R/O/E	6	1	2
Golden20	1867.58	R/E/E/R/R/E/O/R /E/R/R/O/R/R/R/E	9	2	5

R represents reinsertion, E stands for exchange and O represents 2-opt*.

- The exchange neighborhood helps when solutions have overlapping routes like instances Golden_benchmark_9,12,13,16 and 20.

- The 2-opt* neighborhood fits when instances have loose constraints like instance Golden_benchmark_5.

4.7.4 Collective power of multiple neighborhoods

From Table 6, 7 and 8, it is observable that there are two ways in which a neighborhood makes its contribution. First, at a certain stage a neighborhood may be more effective than the others when addressing a certain instance, such as *2-opt** for instance Golden_benchmark_5 and *exchange* for Golden_benchmark_9. By using them together, an algorithm can be more effective for instances with various attributes.

In addition, multiple neighborhoods can cooperate in another way. In the example for the instance Golden_benchmark_5, Thread 3 using exchange neighborhood does not improve the solutions as much as others, but five solutions (underlined numbers in Table 6) improved or modified by Thread 3 enable other threads to find a good solution later. For example, the solution with a value of 6532.27 is found at step 6 by Thread 3 and used as the starting solution at step 10. Subsequently, step by step it goes through 2-opt*, exchange, 2-opt*, exchange and reinsertion neighborhood, leading the search to the best solution with the value of 6460.98 at step 14. Similarly, for the instance Golden_benchmark_9, a solution with the value of 589.97 found by exchange neighborhood at step 3 is used as the starting solution at step 11. Afterwards, it is improved by reinsertion neighborhood 3 times, 2-opt* neighborhood once, exchange neighborhood once and finally reaches a good solution with the value of 584.44. This phenomenon is

also confirmed by the solution paths shown in Table 8, in which all of the solution paths contain all the three neighborhoods.

Thus, all the neighborhoods help finding the best solutions, either improving the solutions more effectively than others or generating intermediate solutions that enable other neighborhoods to find good solutions later. These two aspects may be the main sources of the power of multiple neighborhoods cooperation, which is also one of the major reasons that the proposed method is able to identify high quality solutions.

5 Conclusions

In this paper, we have presented a parallel multi-neighborhood cooperative tabu search algorithm for the capacitated vehicle routing problem. The proposed method exploits the cooperative power of several different tabu search threads and has been tested on the two groups of large scale CVRP benchmarks from the literature. The computational results show that the suggested metaheuristic is effective and competitive in comparison to the best heuristic solution methods from the literature.

In addition, the computational experiments we have performed also reveal some interesting connections. First, the effectiveness of a certain neighborhood is associated with some characteristics of instances it tackles. For example, 2-opt* neighborhood may be more effective for instances loosely constrained while exchange neighborhood is more productive for instances with overlapping routes. Second, in the setting of parallel multiple-neighborhood cooperation, one neighborhood can either contribute by improving the solutions more efficiently than the others or by generating intermediate solutions that enable other neighborhoods find good solutions later. Moreover, the parallel use of multiple neighborhoods facilitates the dynamic adaption of the search to the characteristics of the various instances dynamically. It can thus strengthen the power of the multiple-neighborhood cooperation. Such knowledge can be beneficial for future algorithm design.

The proposed method is simple to implement and can be adapted to address other VRP variants. The modifications required are to adopt problem specific neighborhood operators, the move evaluation criterion and initial solution construction heuristic in order to cater for the different constraints.

Acknowledgements: The authors thank Olli Bräysy for helpful discussions. Thanks are also due to the referees for their valuable comments.

References

R. M. Branchini, V. A. Armentano, and A. Løkketangen. Adaptive granular local search heuristic for a dynamic vehicle routing problem. *Computers & OperationsResearch*, 36:2955–2968, 2009.

T. G. Crainic. Parallel solution methods for vehicle routing problems. In B. Golden,

S. Raghavan, and E. Wasil, editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 171–198, New York, 2008. Springer.

T. G. Crainic and H. Nourredine. Parallel metaheuristics applications. In E. Alba, editor, *Parallel Metaheuristics: A New Class of Algorithms*, pages 447–494, Hoboken, NJ, 2005. John Willey & Sons.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6:80–91, 1959.

B. Dorronsoro, D. Arias, F. Luna, A. J. Nebro, and E. Alba. A grid-based hybrid cellular genetic algorithm for very large scale instances of the cvrp. In *High Performance Computing & Simulation Conference (HPCS)*, pages 759–765, Piscataway, NJ, 2007. IEEE Press.

M. M. Flood. The traveling-salesman problem. *Operations Research*, 4:61–75, 1956.

G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno. Real-time vehicle routing: Solution concepts, algorithms and parallel strategies. *European Journal of Operational Research*, 151:1–11, 2003.

F. Glover and M. Laguna. *Tabu Search*. Kluwer, Boston, 1997.

B. L. Golden, E. A. Wasil, J. P. Kelly, and I. M. Chao. The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In T. Crainic and G. Laporte, editors, *Fleet management and logistics*, pages 33–56, Boston, 1998. Kluwer.

B. L. Golden, S. Raghavan, and E. A. Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, New York, 2008.

C. Groër, B. L. Golden, and E. A. Wasil. A parallel algorithm for the vehicle routing problems. *INFORMS Journal on Computing*, 23:315–330, 2011.

J. Kytöjoki, T. Nuortio, O. Bräysy, and M. Gendreau. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, 34:2743–2757, 2007.

G. Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics*, 54:811–819, 2007.

J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

F. Li, B. L. Golden, and E. A. Wasil. Very large-scale vehicle routing: New test problems, algorithms, and results. *Computers & Operations Research*, 32:1165–1179, 2005.

D. Mester and O. Bräysy. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & Operations Research*, 34:2964–2975, 2007.

Y. Nagata and O. Bräysy. Edge assembly crossover for the capacitated vehicle routing problem. *Networks*, 54:205–215, 2009.

I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. *Annals of Operations Research*, 54:421–452, 1993.

D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34:2403–2435, 2007.

J.-Y. Potvin and J.-M. Rousseau. An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society*, 46:1433–1446, 1995.

M. W. P. Savelsbergh. The vehicle routing problem with time windows: minimizing route duration. *INFORMS Journal on Computing*, 4:146–154, 1992.

P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. PA, Philadelphia, 2002.

P. Toth and D. Vigo. The granular tabu search and its application to the vehicle routing problem. *INFORMS Journal on Computing*, 15:333–346, 2003.