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

Jianyong Jin
Teodor Gabriel Crainic
Arne Løkketangen

December 2010

CIRRELT-2010-54

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

Jianyong Jin[1,2], **Teodor Gabriel Crainic**[1,3*], **Arne Løkketangen**[1,2]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Molde University College, Department of Informatics, Postboks 2110, N-6402 Molde, Norway

[3] Department of Management and Technology, Université du Québec à Montréal, C.P. 8888, succursale Centre-ville, Montréal, Canada H3C 3P8

Abstract. This paper presents a parallel Tabu search algorithm that utilizes several different neighborhood structures for solving capacitated vehicle routing problems. 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 test instance characteristics and uncover the source of the collective power of multi-neighborhood cooperation.

Keywords. Vehicle routing, parallel algorithms, multi-neighborhood, cooperative search, solution pool.

Acknowledgements. Partial funding for this project has been provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), through its Industrial Research Chair and Discovery Grants programs, and by the Fonds de recherche sur la nature et les technologies (FQRNT) through its Strategic Clusters Program.

---

* Corresponding author: Teodor-Gabriel.Crainic@cirrelt.ca

Dépôt légal – Bibliothèque nationale du Québec,
Bibliothèque nationale du Canada, 2010

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 the limited computing time. Exact algorithms have been used to solve the VRP instances with up to 135 customers (Baldacci et al., 2004). 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, some of them use parallel algorithms, in which several (or many) processes work simultaneously on available processors (multi-core CPU or computer clusters) with the common goal of solving a given problem instance. Crainic (2008) describes and discusses 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 in the latest metaheuristics addressing the VRP is to use multiple neighborhoods. In Table 1, we summarize four previously published articles which adopt three or more neighborhoods in their algorithms for solving large scale VRP instances. The authors, the year of publication, the name of the algorithm and the neighborhood structures used in each algorithm are listed. For the details of their methods and neighborhood operators used, we refer to the original papers. 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 and efficient 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 determine their effectiveness for solving the capacitated vehicle routing problems (**CVRP**).

Table 1: Four previous algorithms that use multiple neighborhoods

Authors /Year	**Algorithm**	**Neighborhoods (or other operators) used**
Li et al. (2005)	Variable-length neighbor list record-to-record travel	Reinsertion, Exchange, 2-opt, Perturb a feasible solution.
Kytöjoki et al. (2007)	Variable neighborhood search	2-opt, Or-opt, 3-opt(for intra-route); Reinsertion, Exchange, 2-opt*, CROSS-exchange (for inter-route).
Mester and Bräysy (2007)	Guided local search + evolution strategies	Reinsertion, 2-opt (for intra-route); Reinsertion, Exchange, 2-opt*, Forward Or-exchange, Backward Or-exchange (for inter-route).
Groër et al. (2010)	Record-to-record travel + set covering solver	Reinsertion, Exchange, 2-opt, 3-opt, Or-opt, 3-point move.

The capacitated vehicle routing problem, 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 threads, each using a single neighborhood or a neighborhood combination embedded in a TS. The experiments on 32 large scale CVRP benchmarks demonstrate that the proposed algorithm is effective and efficient. 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 the origin of the collective power of multi-neighborhood cooperation.

The remainder of this paper is organized as follows. In next section our problem solving methodology is introduced. Then Section 3 presents the results of computational tests. Finally, conclusions are drawn in Section 4.

2 Description of the parallel algorithm

In the proposed parallel multi-neighborhood cooperative Tabu search approach, there are three main components introduced below.

First, a set of four basic neighborhoods commonly used in the previous metaheuristics for the VRP are selected to build different search threads. The basic idea of each neighborhood is described as follows.

- Reinsertion (Savelsbergh, 1992) refers to moving a customer node from one position to another. The two positions can be within the same route or in two different routes.
- 2-opt (Flood, 1956) eliminate two edges and add two new edges within the same route.
- Exchange (Osman, 1993) swaps two nodes from two routes.
- 2-opt* (Potvin and Rousseau, 1995) exchanges the head or tail part of two routes.

In this method, 2-opt and reinsertion neighborhoods are used for intra-route operations while inter-route moves are carried out with reinsertion, exchange and 2-opt* neighborhood structures.

Secondly, the selected neighborhoods are applied in a Granular Tabu search (Toth and Vigo, 2003) setting to develop several TS threads. 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 at each iteration, only moves involving one member of the nearest neighbors set will be considered. The size of the set of the nearest neighbors 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). In addition, to explore the solution space more thoroughly, infeasible intermediate solutions are allowed. A penalty mechanism similar to the one described in Toth and Vigo (2003) is used to manage the evolvement between feasible and infeasible search space. Furthermore, to provide different search strategies, each thread utilizes distinct neighborhoods for inter-route operations while all threads use 2-opt and reinsertion neighborhood alternately for intra-route moves. In the subsequent part of this paper, only neighborhoods for inter-route operations in each thread will be discussed and specified for the sake of the simplicity.

For the parallelization aspect, as suggested by Crainic (2008), the cooperative multi-thread search based on a solution pool strategy is applied. In this algorithm, several TS threads cooperate though a solution pool with a common goal of solving a given problem instance as well as possible.

The proposed method consists of two phases. The goal of the first phase is to create a feasible starting solution for the second phase. In the second phase the starting solution is improved by the joint work of four parallel search threads. The framework of the algorithm is showed in Figure 1.

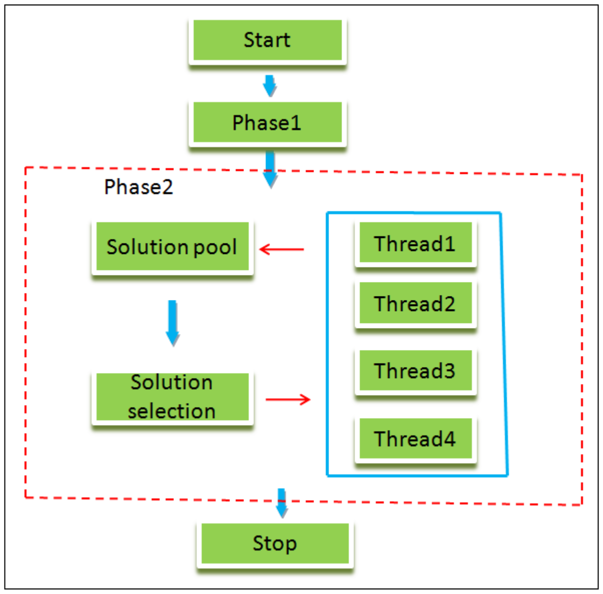

Figure 1: The framework of the proposed method

2.1 The initial solution phase

In the first phase, there are three steps. Step one creates a set of routes with a single customer in each. In step two, the total number of routes is reduced to a preselected level (the minimal number of routes proved feasible before) by repeatedly relocating the customers in a route with the smallest load to other routes which cause the least increase in the sum of the total travel distance and the weighted constraint violations. The solution generated in step two is often infeasible regarding either the capacity constraint or the route length constraint. Thus, in step three, an attempt is made to improve the solution and restore the feasibility by a Tabu search procedure, in which three neighborhoods (reinsertion, exchange and 2-opt*) are used alternately to improve the solution. To restore the feasibility rapidly, a special move selection criterion is used here. Among available moves, those that result in the largest decrease in constraint violations are selected first. When there are no such moves, the least deteriorative moves are also accepted. If there exist the tour length constraints, they are prioritized over the capacity constraints since the latter are relatively easier to restore. The first phase terminates when solution feasibility is attained.

2.2 The improvement phase

In the second phase, the focus is on improving the feasible starting solution. As showed in Figure 1, there are four parallel threads that work together. These threads exchange the best solutions found through a solution pool. The functions of each search thread are depicted in Table 2.

Table 2: The function of each search thread in Phase 2

Component	**Neighborhood used**	**Role**
Thread 1	Reinsertion, 2-opt*.	Main improving thread.
Thread 2	2-opt*.	Assistant improving thread.
Thread 3	Exchange	Assistant improving thread.
Thread 4	Shaking procedure + Improving procedure	Diversifying the search.

Thread 1 utilizes the two neighborhoods (reinsertion and 2-opt*) 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 such a combination is more efficient than a single neighborhood, especially for the instances with tight constraints. For the actual values used, see Section 3.3. 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 trying to improve the given solution. These two neighborhoods can be quite helpful for certain instances. 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 a shaking procedure and an improvement procedure. Firstly, the shaking procedure relocates a certain percentage of nodes of a solution to different positions. The term *shaking strength* is used to refer to this percentage. Half of those nodes are selected from the nodes which are connected to their far neighbors while the other half are selected randomly. After the shaking procedure, the modified solution is improved by the improving procedure, which is a variant of thread 1 (with exchange neighborhood added and different Tabu tenures). The purpose of this thread is to generate diversified solutions to be used by the other threads.

The solution pool, which can be called a solution warehouse as well, keeps the best solutions found by each search thread and sorts these solutions according to 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 fragments). 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 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 will be 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 between two solutions. Here, solutions with a certain amount of different edges are accepted since it is undesirable to start the search from the same solution as where the search stopped. In addition, the solutions in the pool are sorted from the smallest to the largest according to the total travel distance. Moreover, each solution also has a flag that indicates whether it has been previously improved upon. The solutions are checked from the beginning of the solution pool and the first solution which has not been used before and satisfies the difference requirement will be selected for next segment.The reasons for such a configuration are threefold. First, each search thread can start from the best solution found so that all threads can reach the promising search space quickly. Second, all threads are kept in the same area of the search space to intensively search that area. Last, for a given starting solution, the neighborhood that is the most appropriate for generating improvement will always have the opportunity to use this 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 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.

In addition, to speed up the search, the neighborhood exploration and evaluation operators for inter-route moves are parallelized as well. The proposed method has been tested with two sets of CVRP benchmark instances, the results are presented in the next section.

To explore other strategies of utilizing multiple neighborhoods and for comparison purposes, we also tried a serial application variant that utilizes reinsertion, exchange and 2-opt* neighborhood structures in a serial manner (In each iteration a neighborhood is randomly selected according to a certain probability, just like the improvement procedure in Thread 4 mentioned above.) and a single-neighborhood parallel application variant in which there are only three search threads cooperating through the solution pool and each search thread contains a single neighborhood (reinsertion, exchange or 2-opt*) respectively. In the single-neighborhood parallel application variant, the solution pool functions exactly the same way as we discussed above for PMNTS. In addition, these two variants also have the same phase one with PMNTS. They differ only in phase two. These two variants are tested with a subset of CVRP benchmark instances to compare their efficacy. The results are discussed in Section 3 as well.

3 Computational results

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

3.1 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 cores are used by the parallel neighbor generation and evaluation procedures.

3.2 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.

3.3 The algorithm configurations

The parameters of the proposed parallel multi-neighborhood cooperative Tabu search algorithm are shown in Table 3.

Table 3: The parameters of the proposed algorithm

Parameters	**Phase 1**	**Phase 2**			
		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: $\frac{6}{7}$ O: $\frac{1}{7}$	O: 1	E: 1	R: $\frac{5}{7}$ E: $\frac{1}{7}$ O: $\frac{1}{7}$
Nearest neighbors set size	24	10 + random [0, 10]			
Shaking 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			

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

Here, N represents the size of the instance. Tabu tenure is set to be proportional to the size of instances. 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 $\frac{6}{7}$ while the probability of selecting 2-opt* neighborhood is $\frac{1}{7}$. The size of nearest neighbors set is set at 10 plus a uniform random number from the interval [0, 10] at each iteration in phase 2. In Thread 4, 20% of nodes are relocated in the shaking 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 it stops, then the other three threads are terminated as well. 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) during our preliminary computational experiments.

In the serial application variant, there is only one thread using reinsertion, exchange and 2-opt* neighborhoods. The Tabu tenure for each neighborhood is the same in the two phases (R: 0.07N, E: 0.07N and O: 0.135 respectively). The size of the nearest neighbors set and the probability of using each neighborhood are the same as in PMNTS. The search terminates after $60 \times 250\sqrt{N}$ iterations or when there is no improvement for $15 \times 250\sqrt{N}$ iterations.

As for the single-neighborhood parallel application variant, the parameters setting is the same with PMNTS apart from that there is no Thread 4 and only reinsertion neighborhood in Thread 1.

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

In Table 4 we compare the results for the 20 benchmark instances of Golden et al. (1998) against previously published work. From the table, we see the proposed method found new best solutions to 3 problems (numbers in bold font) while the average deviation of our best solutions found 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), Kytjoki et al. (2007) and Gror et al. (2010) using 4 processors, but slightly worse than Mester and Brysy (2007), Nagata and Brysy (2009) and Gror et al. (2010) using 129 processors.

3.5 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 5. 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 Gror et al. (2010) using 129 processors.

Table 4: Comparison of results for the benchmarks of Golden et al. (1998)

Problem	Previous best known	Li et al. (2005)	Psinger and Röpke (2007)	Mester and Bräysy (2007)	Kytöjoki et al. (2007)	Nagata and Bräysy (2009)	Groër et al. (2010) 4pc	Groër et al. (2010) 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		1.13	10.80	24.40	0.02	355.9	5.00	5.00		41.87	

Table 5: Comparison of results for the benchmarks of Li et al. (2005)

Problem	Previous best known	Li et al. (2005)	Psinger and Röpke (2007)	Mester and Bräysy (2007)	Kytöjoki et al. (2007)	Dorronsoro et al. (2007)	Groër et al. (2010) 4pc	Groër et al. (2010) 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		3.20	48.80	104.30	0.10	1830.00	5.00	5.00		160.43	

3.6 Observations on neighborhood effectiveness

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

3.6.1 Both serial and parallel strategy have advantages

In Table 6, the average results of the two variants (SNP in Table 6 represents single neighborhood parallel variant) for a subset of test instances are compared together with PMNTS. To distinguish the test instances, the tightness of the constraints, which are measured with the values of $\frac{average-route-load}{vehicle-capacity}$ and $\frac{average-tour-length}{tour-length-constraint}$ respectively, are provided in the table. By comparing the outcome, we see that the PMNTS algorithm outperforms the serial variant in both solution quality and wall-clock time. As for the single neighborhood parallel variant, it seems that it exceeds the serial variant often on the instances with loose constraints such as instance 5 and 16. For those tightly constrained instances like instance 1 and 20, its performance is even worse than the serial variant. This observation indicates both using multiple neighborhoods in serial manner and parallel manner have advantages.

Table 6: Comparison of results of the other two variants with PMNTS

Problem	Previous	Serial variant		SNP variant		PMNTS		Constraints tightness	
	best	Obj	Time	Obj	Time	Obj	Time	Capacity	Tour length
	known		(min)		(min)		(min)	%	%
1(240)	5623.47	5636.77	26.15	5667.90	13.49	5627.54	16.54	97.0	96.1
4(480)	13592.88	13790.27	100.37	13894.88	63.08	13666.84	87.03	96.0	85.2
5(200)	6460.98	6489.97	21.41	6473.53	9.12	6464.40	10.47	88.9	71.8
8(440)	11643.90	11797.88	88.74	11774.16	45.38	11725.82	86.68	97.8	97.1
9(255)	579.71	586.51	29.70	585.27	17.88	583.15	18.70	95.9	N/A
12(483)	1102.76	1120.62	105.93	1114.21	64.93	1112.44	72.84	98.4	N/A
13(252)	857.19	872.00	17.26	869.95	12.32	864.45	18.82	96.7	N/A
16(480)	1613.66	1647.00	60.81	1638.92	40.90	1632.88	55.48	96.7	N/A
17(240)	707.76	709.80	14.81	714.68	11.40	708.46	15.83	98.2	N/A
20(420)	1818.25	1839.13	42.44	1871.28	33.84	1830.10	49.62	99.5	N/A
Aver. deviation %		1.15		1.33		0.56			
Time min			50.76		31.23		43.20		

3.6.2 An example where 2-opt* neighborhood is effective

In Table 7, an example is provided in which all the improving related steps that the single-neighborhood parallel application variant 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 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 7: 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 7, we can see 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 with reinsertion neighborhood often provides the best during the late steps. 2-opt* neighborhood 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 tour 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 tour 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 2 visualizes the features of different solutions to this instance.

3.6.3 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 the single-neighborhood parallel application variant for the instance Golden_benchmark_9 is instantiated in Table 8 to demonstrate such cases.

From Table 8, we can see exchange neighborhood has found the best solutions among

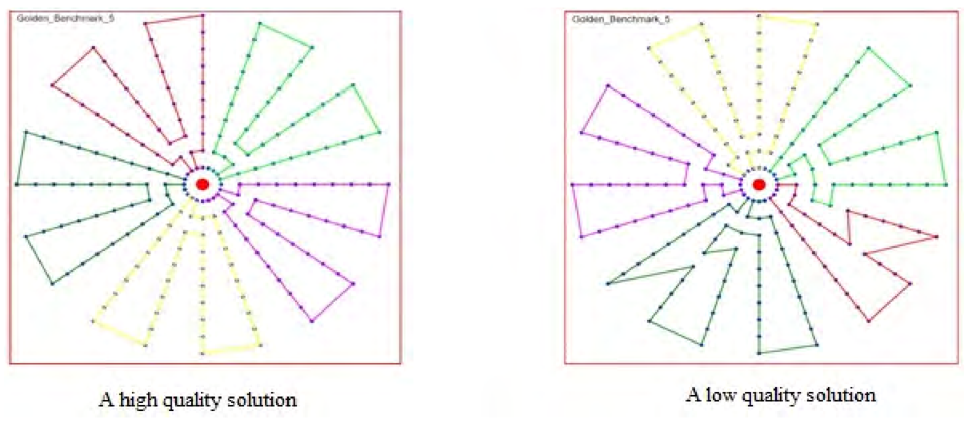

Figure 2: Solution features of Golden_benchmark_5

Table 8: 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

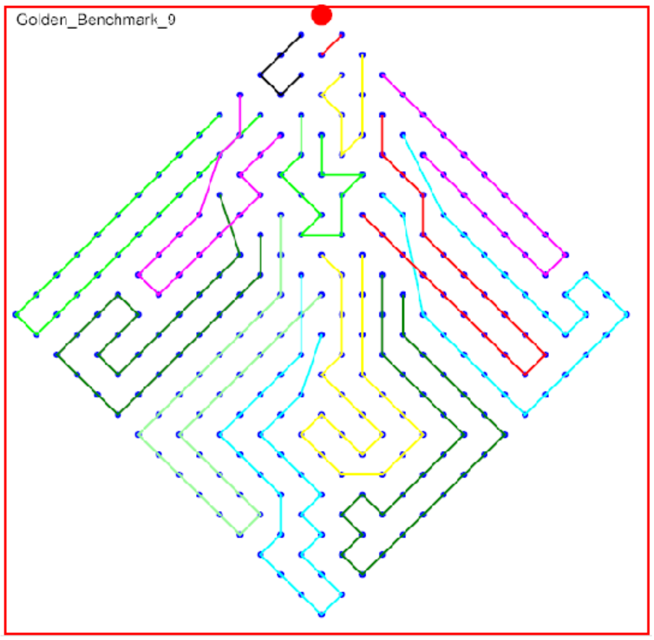

Figure 3: Solution features of Golden_benchmark_9

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 3, from which it is noticeable that the solution has routes overlapping each other. Such a feature facilitates exchange neighborhood to find good moves.

3.6.4 General trend of neighborhood effectiveness

To find the general trend of neighborhood effectiveness, the search steps that the single-neighborhood parallel application variant took to identify the best solution for ten instances are summarized in Table 9. 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 9, one may notice the following trends.

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

Table 9: 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

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

- 2-opt* neighborhood fits when instances have loose constraints like instances in Golden_benchmark_5 and 16.

3.6.5 Collective power of multiple neighborhoods

From Table 7, 8 and 9, 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 efficient for instances with various attributes.

In addition, multiple neighborhoods can cooperate in another way. In the example for the instance Golden_benchmark_5, Thread3 using exchange neighborhood does not improve the solutions as much as others, but five solutions (underlined numbers) 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 9, 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 should be the main sources of the power of multiple neighborhoods cooperation, which is also the major reason that the proposed method is able to identify high quality solutions.

4 Conclusions

In this paper, we have presented a parallel multi-neighborhood cooperative Tabu search algorithm for the capacitated vehicle routing problems. The proposed method exploits the cooperative power of several 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 facts. First, the effectiveness of a certain neighborhood is associated with the 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. Last, both using multiple neighborhoods in serial manner and parallel manner have advantages. Such knowledge can be beneficial for future algorithm design.

Acknowledgments

The authors thank Dr. Olli Bräysy for helpful discussions and Mr. François Guertin of the Réseau Québécois de Calcul de Haute Performance (RQCHP) for his assistance and insightful comments regarding parallel computing and the utilization of the equipment of the RQCHP.

While working on this project, the second author was the NSERC Industrial Research Chair in Logistics Management, ESG UQAM, and Adjunct Professor with the Department of Computer Science and Operations Research, Université de Montréal, and the Department of Economics and Business Administration, Molde University College, Norway. Partial funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Industrial Research Chair and Discovery Grants programs, and by the Fonds de recherche sur la nature et les technologies of the Province of Quebec through its Strategic Clusters program.

References

R. Baldacci, E. A. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation. *Operations Research*, 52:723–738, 2004.

R. M. Branchini, V. A. Armentano, and A. L. kketangen. 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*, 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.

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

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

B. L. Golden, E. A. Wasil, J. P. Kelly, and I.-M. 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. *The Vehicle Routing Problem: Latest Advances and New Challenges*. New York, 2008. Springer.

C. Groër, B. L. Golden, and E. A. Wasil. A parallel algorithm for the vehicle routing problems. *INFORMS Journal on Computing*, Forthcoming, 2010.

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. R. 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. *The vehicle routing problem.* SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002. PA.

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.