Computers & Operations Research 44 (2014) 33–41

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

A cooperative parallel metaheuristic for the capacitated vehicle routing problem

Jianyong Jin [a], Teodor Gabriel Crainic [b,*], Arne Løkketangen [a]

[a] Molde University College, Specialized University in Logistics, N-6411, Molde, Norway
[b] School of Management, UQAM & Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, Montréal, QC, Canada

ARTICLE INFO

Available online 25 October 2013

Keywords:
Vehicle routing
Parallel metaheuristic
Cooperative search
Solution clustering

ABSTRACT

This paper introduces a cooperative parallel metaheuristic for the capacitated vehicle routing problem. The proposed metaheuristic consists of multiple parallel tabu search threads that cooperate by asynchronously exchanging best-found solutions through a common solution pool. The solutions sent to the pool are clustered according to their similarities. The search history information identified from the solution clusters is applied to guide the intensification or diversification of the tabu search threads. Computational experiments on two sets of large-scale benchmark instance sets from the literature demonstrate that the suggested metaheuristic is highly competitive, providing new best solutions to ten of those well-studied instances.

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

In recent years, cooperative parallel metaheuristics have increasingly been used for solving a variety of difficult combinatorial problems [1]. Such parallel metaheuristics usually use multiple processes (threads) working simultaneously on available processors, with varying degrees of cooperation, to solve a given problem instance. The rationale behind this phenomenon may be twofold. First, it has been demonstrated that such parallel algorithms are capable of both speeding up the search and improving the robustness (ability of providing equally good solutions to a large and varied set of problem instances) and the quality of the solutions obtained [2]. Second, parallel computing resources have become increasingly available with the advent of computer clusters and multi-core processors. The computer clusters usually consist of a set of identical computers that run standard operating systems and are connected to each other through high speed networks. Many universities nowadays possess such computer clusters. In addition, many laptops and desktop computers today use dual- or quad-core processors. Thus, using parallelism has become an advantageous and practical option. For a detailed introduction to parallel metaheuristics, we refer to the book of Alba [3] and the survey papers of Crainic [2] and Crainic and Toulouse [4].

The capacitated vehicle routing problem (CVRP), the classical version of the vehicle routing problem (VRP), aims to determine the minimum total cost routes for a fleet of homogeneous vehicles to serve a set of customers. The CVRP can be defined on a graph $G=(N,E)$, where $N=\{0,\ldots,n\}$ is a vertex or node set and $E=\{(i,j): i,j\in N\}$ is an edge 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 node 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 any route must not exceed the vehicle capacity Q. Some CVRP instances may have an additional route duration limit constraint, restricting the duration (or length) of any route to a preset bound D. A detailed introduction to the CVRP and its solution methods can be found in the book of Toth and Vigo [5], and the survey paper of Laporte [6]. Even though a large number of solution methods have been proposed in the literature during the last 50 years, it still remains computationally challenging to quickly produce high-quality solutions to large scale CVRP instances.

The purpose of this paper is to present a cooperative parallel metaheuristic that takes advantage of modern parallel computing resources to address large scale CVRP instances. The proposed algorithm incorporates multiple tabu search threads which cooperate by asynchronously exchanging the best-found solutions through a common solution pool, and includes several novel features. Intensification and diversification of the tabu searches are based on solution clustering. Four variants of the reinsertion neighborhood are applied and unfeasible solutions may also be sent to the solution pool. These features are clearly different from previous work (e.g., [7,8]) and largely contribute to the high performance of the proposed metaheuristic. The computational experiments on two sets of

* Corresponding author. Tel.: +1 514 343 7143; fax: +1 514 343 7121.
E-mail address: TeodorGabriel.Crainic@cirrelt.ca (T.G. Crainic).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.10.004

large scale CVRP benchmark instances demonstrate that the suggested metaheuristic can quickly produce solutions to benchmark problems that are highly competitive with the best solutions reported in the literature. New best solutions to 10 out of the 32 instances have been identified.

The remainder of this paper is organized as follows. In the next section the description of the proposed metaheuristic is presented. Then Section 3 reports the computational results. Concluding remarks are given in the last section.

2. Description of the cooperative parallel metaheuristic

In the proposed cooperative parallel metaheuristic (CPM), illustrated in Fig. 1, multiple tabu search (TS) threads are run in parallel to address a given CVRP instance. Some of the TS threads are designated to concentrate on intensification while the others are assigned to pursue diversification. These threads communicate asynchronously through a common solution pool.

The general scheme of CPM is displayed in Algorithm 1. During the search process, the solution pool receives solutions sent from the search threads. Whenever a solution is received from a search thread, the pool performs the clustering, selects a solution, and sends it back to the same thread. Each of the TS threads carries out its search independently and periodically the search halts and exports its best-found solution. It then receives a solution from the pool and resumes its search from this solution. The detailed description of the solution pool and the TS threads is provided in Sections 2.1 and 2.2 respectively.

The termination of CPM can be controlled in two ways. In the first setting (identified as TC1), termination is triggered by the first TS thread. The metaheuristic terminates after the thread runs for a certain number of iterations. In the other setting (called TC2), the metaheuristic terminates once the solution pool receives a certain number of non-improving solutions consecutively. A solution is regarded as non-improving when it is unfeasible or its value is not better than that of the current best feasible solution in the pool.

Algorithm 1. CPM

```
Initialize TS threads and the solution pool;
while termination condition not met
   Solution pool
      Receives solutions;
      Clusters solutions;
      Selects and sends solutions back;
   Each TS thread asynchronously
      Performs the search;
      Sends best found solution to the solution pool;
      Receives new solution to start from;
end while
Return best feasible solution.
```

intensification
s làm tăng cao lên, s tăng cng
s làm cho mãnh lit, s làm cho d di
s làm sâu sc thêm, s làm mnh thêm
(nhip nh) s làm ni thêm

In terms of the taxonomy of Crainic and Hail [9] for parallel metaheuristics, CPM fits into the *pC/KC/MPDS* classification. The first dimension *pC* indicates that the global search is controlled by multiple cooperative threads. The second dimension *KC* stands for knowledge collegial information exchange and refers to the fact that multiple threads share information asynchronously and knowledge is created from the exchanged information to guide the cooperating threads. The last dimension *MPDS* indicates that multiple search threads start from different points in the solution space and follow different search strategies.

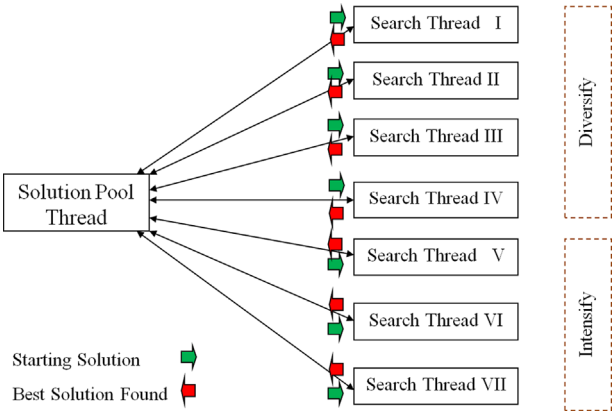

Fig. 1. Framework of CPM.

2.1. Solution pool

To explore a search space effectively and efficiently, a metaheuristic approach should be able to both intensively investigate the areas of the search space displaying high quality solutions, and to move to unexplored areas of the search space when necessary. These goals are usually achieved by intensification and diversification mechanisms of the metaheuristic [10]. Glover and Laguna [11] highlight that intensification is to carefully search the neighborhood of elite solutions while diversification encourages the search process to generate solutions that differ from those seen before. A solution clustering approach is used in CPM to implicitly identify common features of solutions and collect search history information, which then provides a good basis for selecting promising search areas for intensification and less explored areas for diversification.

During the whole search process, the solutions sent to the solution pool from the search threads are dynamically clustered into groups according to their similarity. For the CVRP, similarity can be measured in terms of the number of edges solutions have in common. Solutions kept in the solution pool can then be grouped into clusters where all solutions belonging to the same cluster have a given number of edges in common. Each cluster can thus approximately represent a region of the search space that CPM has explored. The features of the solutions in a cluster, such as the number of solutions and the quality of the solutions, can indicate how thoroughly a search region has been explored and how promising it may be. Such search history information is used to guide the starting solution selection for the TS threads so that they can pursue intensification or diversification effectively.

The solution clustering approach has been applied by Voß [12] for the quadratic assignment problem. In his algorithm, a small number of elite solutions previously found are stored by a clustering approach and are used as the starting solutions for the intensification phases. In CPM, the solution clustering approach is applied differently in three aspects. First, all solutions sent to the pool are clustered, regardless of their quality. Second, the solutions are clustered for both intensification and diversification purposes. Third, the actual clustering mechanism is different.

2.1.1. Solution clustering

Clustering is often defined as the process of grouping of a collection of patterns into dissimilar segments or clusters based on a suitable notion of closeness or similarity among these patterns. In CPM, solutions are grouped into clusters based on their similarity. A cluster, in this context, refers to a collection of solutions that are similar. All solutions sent to the solution pool are clustered.

To support solution exchange and information extraction from the clusters, a set of components are implemented for each cluster:

- *Feasible solution list*: the feasible solutions assigned to the cluster.
- *Unfeasible solution list*: the unfeasible solutions assigned to the cluster.
- *Edge residence counter* for all edges. The residence counter of an edge is defined as the number of feasible solutions containing that edge that have been assigned to the cluster. Since the objective of the search is to identify high-quality feasible solutions, only feasible solutions are used to compute the edge residence counter.
- *Feasible solution counter*: the number of feasible solutions assigned to the cluster.
- *Unfeasible solution counter*: the number of unfeasible solutions assigned to the cluster.
- *Average feasible solution value* of all feasible solutions in the cluster.
- *Average unfeasible solution value* of all unfeasible solutions in the cluster.

Whenever a solution enters a cluster, the components of the cluster are updated. In each cluster, duplicate solutions are eliminated and the lists of feasible and unfeasible solutions are sorted in ascending order according to the solution value. In addition, the clusters are sorted in an ascending order according to the average feasible solution value. When there are only unfeasible solutions in a cluster, we replace the average feasible solution value with the average unfeasible solution value for sorting.

To determine whether a solution is similar to the solutions in a cluster, the similarity between the solution and the cluster is calculated according to Eq. (1).

$$Similarity = \frac{\sum_{(ij)\in E} n_{ij} \times X_{ij}^{s}}{FSC \times \sum_{(ij)\in E} X_{ij}^{s}}, \quad (1)$$

where

- n_{ij}: the residence counter of edge (i,j) of the cluster;
- X_{ij}^{s}: 1 if edge (i,j) appears in solution s, 0 otherwise;
- FSC: the feasible solution counter of the cluster;
- $\sum_{(ij)\in E} X_{ij}^{s}$: the number of edges in solution s.

The advantage of computing the similarity in such a way is twofold. First, it does indicate how many common edges a solution shares with the solutions in a cluster. Moreover, it avoids the heavy computational load of calculating the similarities between a solution and every solution in the cluster as other clustering approaches do.

A solution can be placed into a cluster only if the similarity between the solution and the cluster is larger than a minimum value. This value is termed the minimal similarity requirement. When a solution is sent to the solution pool, three possibilities exist. Initially, there is no existing cluster, a cluster is created and the solution is directly placed into the cluster. When there are existing clusters, the solution is compared with the first cluster in the pool. If the similarity requirement is satisfied, it is put into the cluster. Otherwise, it is compared with the next cluster in the sorted order. The comparison may continue until the solution is placed into a cluster or it does not satisfy the similarity requirement with any existing cluster. Under such a circumstance, a new cluster is created and the solution is put into the new cluster.

A pair of status flags is attached to each solution that enters a cluster to signal whether it has been used or not. The two flags are used for intensification and diversification respectively. After a solution is selected and sent to an intensification TS thread, its status flag for intensification is set accordingly. Likewise, the status flag for diversification is set once a solution is sent to a diversification TS thread. By setting these flags, each solution can be usually selected and sent to each type of search thread only once.

Three parameters are used to control the clustering process. The first one is the minimal similarity requirement, called *minSim*, which controls the number of common edges the solutions in a cluster share. The second one is the maximal number of clusters, *maxNC*, which restricts the number of clusters in the solution pool, as too many clusters may slow down the clustering process. Whenever a new cluster is created, the clustering procedure checks the number of existing clusters. If there are more clusters than allowed, a cluster that does not contain any feasible solution or has the largest average feasible solution value is eliminated. The last parameter is the maximal number of solutions in a cluster, *maxNS*. When there are more solutions than allowed, the worst solution in terms of the solution value is removed. It is essential to restrict the number of the clusters and the number of solutions in each cluster for the sake of efficiency, especially when a large number of threads are employed.

eliminated: loai tru

2.1.2. Solution selection for intensification threads

For a cluster that contains mainly high-quality solutions, indicated by the average feasible solution value of the cluster, the search region represented by the cluster is usually worth further intensive investigation. In CPM, the cluster having the lowest average feasible solution value is assigned as the target of the TS threads that pursue intensification. These search threads only receive starting solutions from this best cluster so that the neighborhoods of high-quality solutions can be thoroughly investigated. During the entire search process, this best cluster may dynamically be replaced by newly emerged clusters that have lower average feasible solution values than the current one. In this way, the intensification search threads are always targeting the vicinity of the current best solutions.

Whenever an intensification thread needs a starting solution, the solutions in the best cluster are checked. The intensification flag of each solution is examined one at a time, starting from the solution with the lowest solution value, following the sorted order. The first unused solution is selected. When there are no unused solutions in the best cluster, a solution is randomly selected from the cluster. This selected solution is sent to the search thread.

If there are unfeasible solutions in the best cluster, they are examined and selected first. The reason for this decision is that preliminary experiments showed that there are usually many more feasible solutions than unfeasible solutions in the clusters. The unfeasible solutions will seldom be selected if the feasible solutions are checked first.

2.1.3. Solution selection for diversification threads

The search regions that have been less thoroughly explored can be indicated by the number of feasible solutions that have been put into the clusters. The fewer feasible solutions have been put into a cluster, the less thoroughly the region has been searched. Since the tabu search threads seeking diversification are expected to concentrate mainly on less explored search regions, they receive starting solutions only from a cluster whose feasible solution counter is below a threshold. We term this threshold the diversification threshold. When the feasible solution counter of a cluster exceeds the diversification threshold, the cluster is disregarded while selecting solutions for the diversification threads.

Whenever a diversification thread requires a starting solution, the solution selection procedure starts with the cluster with the lowest average feasible solution value in the solution pool. If the feasible solution counter of the cluster is below the diversification threshold, the solutions in the cluster are then examined. If an unused solution is found, this solution is sent to the search thread, otherwise, the next cluster following the sorted order is checked. The examination continues until an unused solution is found or all available clusters have been checked. When an unused solution is not found after examining all clusters, a solution is randomly selected. When examining the solutions in a cluster, the unfeasible solutions are checked and selected first, as for intensification threads.

2.2. The Tabu search threads

The general structure of the TS threads included in CPM is displayed in Algorithm 2. The rest of the sub-section is dedicated to detailing the common features and differences between these TS threads.

Algorithm 2. Tabu search

1: Construct s_i, set $s_f^* = s_i$, $s_{inf}^* = s_i$, $s = s_i$;
2: Initialize tabu lists and penalty multipliers;
3: **while** termination condition not met **do**
4: Select a neighborhood and *LS*;
5: Generate and evaluate neighboring solutions;
6: Select a neighboring solution $\overline{s}$ minimizing $F(\overline{s})$ and is non-tabu or satisfies the aspiration criterion,and set $s = \overline{s}$;
7: Declare the attributes of the reverse moves tabu for *tt* iterations;
8: Refine the routes modified;
9: Update s_f^* and s_{inf}^*;
10: Update penalty multipliers;
11: If reaching the iteration limit, halt and exchange solutions with the solution pool; reset s_f^*, s_{inf}^*, s, and tabu lists;
12: **end while**

2.2.1. The common features of the tabu search threads

The TS threads included in CPM are developed on the basis of the granular tabu search introduced by Toth and Vigo [13]. Their main features are described in the following.

The initial solution: The initial solution of each TS thread is constructed by using the parameterized Clarke–Wright algorithm described in Yellow [14] with a randomly generated shape parameter. The range for the shape parameter is set to (0.5, 2) as suggested in Groër et al. [15].

Objective function and constraint relaxation mechanism: To explore the solution space more thoroughly, unfeasible intermediate solutions are allowed. To this end, 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 $D(s)$ stand for the total violations of the 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. To be precise, every 10 iterations, α is set to $\alpha/2$ if all the previous 10 visited solutions were feasible with respect to the capacity constraints, it is set to 2α if all were unfeasible, and left unchanged otherwise. The adjustment rule for β is similar. The two multipliers are initially set to 1. The neighboring solutions, both feasible and unfeasible, generated during the search process are evaluated in terms of the augmented objective function.

Inter-route neighborhood structures: Three neighborhood structures for inter-route operations, namely reinsertion [16], 2-opt* [17], and CROSS-exchange [18], which are commonly used in the previously published metaheuristics for the CVRP, are included in each TS thread. One of them is randomly selected every TS iterations and each of the neighborhoods has equal probability to be selected.

To speed up the search, the granular neighborhood reduction technique applied in Jin et al. [7] is adopted in CPM. Let $R(u)$ stands for the route containing node u in a given solution, and (u,x) be the partial route from node u to node x. Define N_u as the set of the nearest neighbors of customer u. Assume node v is a member of N_u and $R(v) \neq R(u)$. Each neighborhood is generated in the following way:

- Reinsertion: For each customer u, for each v, reinsert u right after v.
- 2-opt*: Let x be the successor of u in $R(u)$ and y be the successor of v in $R(v)$. For each customer u, for each v, replace (u,x) and (v,y) by (u,v) and (x,y).
- CROSS-exchange: The procedure introduced in Taillard et al. [18] is adopted. To reduce the computational effort, one restriction is imposed. For a route R_1, only a couple of routes are chosen for neighborhood generation. Those routes are selected in the following way. First, a node u is randomly selected from the middle part of route R_1. Then the routes which contain at least one customer node belonging to N_u are identified. Only these routes are used to form route pairs with R_1 for neighborhood generation.

The size of the nearest neighbors set is randomly chosen within a certain range at each iteration.

Four types of reinsertion strategies: The reinsertion neighborhood structure for the inter-route improvement of the TS threads, is implemented in four distinct ways. The main idea is to partition the customer nodes into groups according to their distance to the depot, and each group should have an approximately equal number of nodes. At each iteration, neighborhood generation and move selection are carried out separately for each group. In such a way, the frequency of modifying the route structures (edges) distant from the depot can be increased and brought closer to that of the edges located close to the depot. An example is shown in Fig. 2, where the small square stands for the depot and the dots represent customer nodes. The customer nodes are divided into two groups, the dots inside the big circle belong to the first group while the dots outside the big circle constitute the other group. Nevertheless, the nearest neighbors (the dots inside the small circle) of a customer do not need to be in the same group as the customer.

The first type of reinsertion strategy is to put all customer nodes in one group, and at each iteration only one move is performed. For the second strategy, the customer nodes are partitioned into two groups and two moves are carried out at each iteration, one from each partition. Likewise, for the other two strategies, the customer nodes are divided into 3 or 4 groups and 3 or 4 moves are performed at each iteration respectively. These four types of strategies are termed Type 1 reinsertion, Type 2 reinsertion, Type 3 reinsertion and Type 4 reinsertion, accordingly. For a given TS thread, only one type of reinsertion is applied.

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 in the current search.

The tabu list is neighborhood dependent. The tabu tenure *tt* of each neighborhood is set to be proportional to the number of nodes in the instance. For reinsertion, if u is relocated, u is declared

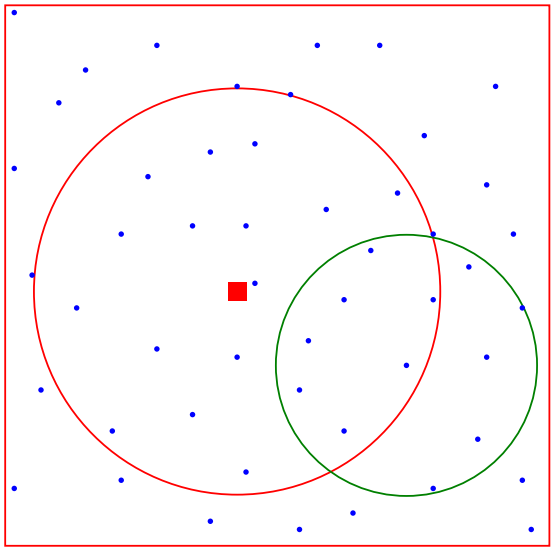

Fig. 2. Customer nodes partition.

tabu for tt iterations and any move relocating u cannot be performed unless it satisfies the aspiration criterion. For CROSS-exchange move swapping route segments (X_1, Y_1) and (X_2, Y_2), nodes X_1 and X_2 are declared tabu and any move involving the two nodes cannot be performed unless it satisfies the aspiration criterion. For 2-opt* move adding edges (u,v) and (x,y), nodes u, v and y are declared tabu, any move involving any one of these three nodes is 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 improvement procedure. The procedure consists of two simple heuristics developed by implementing 2-opt [19] and reinsertion [16] neighborhood structures in a local search setting. The two heuristics are applied to a route alternately. The procedure terminates when no improvement can be found.

Solution exchange: The TS threads halt and exchange solutions with the solution pool periodically. Each TS thread exports its best-found solution and receives a new solution to resume the search. Each search thread decides when to exchange solutions with the solution pool according to its own search trajectory, the communication is asynchronous. No direct communication takes place between the search threads.

The search effort between two solution exchanges is termed a search period. During a search period, if the best feasible solution that a TS thread has found is better than its starting solution, this solution is sent to the pool, otherwise, the best unfeasible solution that the thread has found is sent to the pool. The rationale behind the decision to exchange unfeasible solutions is that unfeasible solutions generated by one thread may be improved by another thread so that better feasible solutions can be found.

The TS threads can stop and exchange solutions with the solution pool after either running for a certain number of iterations or failing to find improving feasible solutions for a certain number of iterations. The first stopping mechanism is identified SM1 and the second one SM2. The TS threads for intensification and diversification can adopt the same (all use SM1 or SM2) or different (some use SM1 and the others use SM2) stopping mechanisms. Several settings are compared in Section 3.3.

Solution representation and transformation: To speed up the computation, the solutions are stored in a data structure of four arrays, namely next-array, pred-array, start-array, and end-array. The first two arrays keep the successor and the predecessor of each node while the other two record the first customer and the last customer of each route. Using this structure, changes to a solution can be performed very quickly. The detailed description of this application can be found in Kytöjoki et al. [20] and Groër et al. [21]. On the other hand, to simplify the information exchanged between the search threads and the solution pool, the solutions are transformed to a giant tour format (without route delimiters) before they are sent to the solution pool. When a TS thread receives a solution from the pool, the giant tour is transformed back to the four-array format with a split algorithm presented by Prins [22]. This split algorithm considers both vehicle capacity and route length constraints, and all solutions are feasible after split.

To facilitate the solution clustering and sorting in the pool, the feasibility and the objective function value of each solution are required. To this end, these two attributes are attached to the giant tour during the transformation and are exchanged together. In the pool, each solution is stored as an augmented giant tour.

2.2.2. The differences between the tabu search threads

To explore the search space effectively and efficiently, some of the TS threads are designated to concentrate on intensification while the others are assigned to pursue diversification. The intensification threads are implemented with smaller tabu tenures than the threads seeking diversification. Additionally, each TS thread uses only one of the four types of reinsertion neighborhoods.

3. Computational results

In this section we describe the experimental platform, the test data sets, the algorithm configurations and compare the experimental results against the results of the state-of-the-art methods and the best known solutions (BKS) reported in the literature. The current best known results have been updated to include the new best solutions identified by Groër et al. [15], Jin et al. [8] and Vidal et al. [23].

The analysis of the impact on the performance of several algorithmic components and the evaluation of the parallel speedup are provided in this section as well.

3.1. Experimental platform and implementation issues

The proposed metaheuristic is implemented in C++ and uses the message passing interface (MPI) for the inter-processor information exchange. The results were obtained by running the algorithm on a compute cluster in which each node consists of two AMD 6172 processors with 12 cores at 2.1 GHz.

The basic configuration of CPM has eight threads. Among them, one thread is used for the solution pool, four TS threads using Type 1, 2, 3 and 4 reinsertion respectively are for diversification and the remaining three TS threads using Type 1, 2, and 3 reinsertion respectively are for intensification. These seven TS threads are regarded as the basic search threads. When more processors are employed by CPM, the seven basic search threads can be duplicated and run on the available processors. The standard configuration of CPM (called CPM standard) utilizes 24 threads with one for the solution pool, 14 for diversification and 9 for intensification.

3.2. The test data sets

The computational tests were carried out on the CVRP benchmarks of Golden et al. [24] and Li et al. [25]. The 20 benchmark instances of Golden et al. [24] have 200–483 customers. The first eight instances also have route length restrictions. The 12 benchmark instances of Li et al. [25] have 560–1200 customers and route length restrictions. For each instance under each experimental scenario, CPM was executed 10 times with different random seeds.

The average result and the best result of these 10 runs are reported.

3.3. Evaluating the search stopping mechanisms

As mentioned in Section 2, the overall search of CPM can be terminated according to two conditions, TC1 and TC2. For the TS threads, there are two stopping mechanisms (SM1 and SM2) for deciding when to exchange solutions with the solution pool. In addition, some of the TS threads are designated to concentrate on intensification while the others are assigned to pursue diversification. Considering these three aspects, six ways of controlling the search of CPM are compared. The settings of the six variants are shown in Table 1.

In general, TC1 and SM1 can explicitly control the search effort while TC2 and SM2 may stop the search dynamically according to the progress of the TS threads. Preliminary testing gave no significant difference among the six variants in terms of the solution quality and the search time. Therefore we choose variant 1 to perform the remaining computational experiments since it allows us to explicitly control the search effort.

Table 1
Search stopping mechanisms.

Variant	1	2	3	4	5	6
Overall search	TC1	TC1	TC1	TC2	TC2	TC2
Diversification TS threads	SM1	SM2	SM1	SM1	SM2	SM1
Intensification TS threads	SM1	SM2	SM2	SM1	SM2	SM2

Table 2
Parameter values for CPM.

Parameter	Value
Tabu tenure of reinsertion, 2-opt* and CROSS-exchange	$0.03\|N\|$ for diversification threads $0.01\|N\|$ for intensification threads
Nearest neighbors set size	$(10 + \text{random}[0, 10])$
Solution exchange	$200 \times \sqrt{\|N\|}$ iterations.
minSim	0.7
maxNC	100
maxNS	300
Diversification threshold	100
Termination condition	$150\,000 \times \sqrt{\|N\|}$ for $\|N\| < 500$ $30\,000 \times \sqrt{\|N\|}$ for $\|N\| > 500$

$|N|$ represents the instance size.

3.4. Algorithm calibration

The parameters of CPM were selected according to the computational results of preliminary experiments on the benchmarks of Golden et al. [24]. A number of different alternative values were tested and the ones selected are those that gave the best computational results concerning both the quality of solutions and the computational times needed to achieve these solutions. The selected parameter values are given in Table 2.

3.5. Results for the benchmarks of Golden et al. [24]

In Table 3, we compare the results for the 20 benchmark instances of Golden et al. [24]. In the table, the first column describes the instances (instance number and number of nodes). The second column lists the best known solutions previously reported in the literature. The third and fourth columns provide the best results presented by Groër et al. [15] and Vidal et al. [23]. The remaining columns give the average results, average wall-clock time, standard deviations, and best results of CPM standard. The first of the three last rows presents the average deviation of all instances from the best known solutions. The second last row provides the average wall-clock time per instances for a single run. The last row shows the number of runs performed for each instance in each algorithm.

From the table, we see that CPM standard has found new best solutions to seven instances (numbers in bold font) while the average deviation of the best results from the best known solutions is 0.00%. In terms of this metric, the best results generated by

Table 3
Comparison of results for benchmarks of Golden et al. [24].

Instances	Previous best known	Groër et al. [15] 129p	Vidal et al. [23]	CPM standard			
				Aver.	Time (min)	SD	Best
1(240)	5623.47	5623.47	5623.47	5623.65	22.05	0.38	5623.47
2(320)	8404.61	8435.00	8404.61	8434.78	34.22	14.45	8405.81
3(400)	11 036.22	11 036.22	11 036.22	11 036.22	44.64	0.00	11 036.22
4(480)	13 592.88	13 624.52	13 624.52	13 620.30	60.87	10.90	**13 590.00**
5(200)	6460.98	6460.98	6460.98	6460.98	15.83	0.00	6460.98
6(280)	8400.33	8412.90	8412.9	8404.06	26.76	4.97	8400.33
7(360)	10 102.70	10 195.59	10 102.70	10 134.93	39.01	11.48	10 107.49
8(440)	11635.30	11 649.89	11 635.30	11 635.34	54.61	0.00	11 635.34
9(255)	579.71	579.71	579.71	580.04	19.43	0.29	579.71
10(323)	736.26	737.28	736.26	737.16	28.82	0.46	**735.66**
11(399)	912.84	913.35	912.84	912.72	41.33	0.28	**912.03**
12(483)	1102.69	1102.76	1102.69	1103.20	58.29	1.28	**1101.50**
13(252)	857.19	857.19	857.19	858.57	18.06	1.20	857.19
14(320)	1080.55	1080.55	1080.55	1080.55	25.08	0.00	1080.55
15(396)	1337.92	1338.00	1337.92	1340.13	36.33	1.38	**1337.87**
16(480)	1612.50	1613.66	1612.50	1614.73	48.14	1.80	**1611.56**
17(240)	707.76	707.76	707.76	707.80	16.39	0.07	707.76
18(300)	995.13	995.13	995.13	998.90	25.01	0.96	997.58
19(360)	1365.60	1365.60	1365.60	1366.12	32.60	0.37	1365.60
20(420)	1818.25	1818.25	1818.32	1819.76	41.93	1.10	**1817.89**
Aver. deviation (%)		0.10	0.02	0.11			0.00
Time (min)		5.00	58.56		34.47		
Runs per instances		5	10	10			10

CPM standard are better than those of Groër et al. [15] and Vidal et al. [23]. The wall-clock time required by CPM standard appears shorter than what was used in Vidal et al. [23] and longer than for Groër et al. [15].

3.6. Results for the benchmarks of Li et al. [25]

The results for the 12 benchmark instances of Li et al. [25] are presented in Table 4. The format of this table is identical to the one of Table 3. For this set of instances, CPM standard has found new best solutions to three instances (numbers in bold font). In terms of the average deviation from the best known solutions, both the average and best results of CPM standard excel the best results of Mester and Bräysy [26] and Groër et al. [15]. The wall-clock time required by CPM standard turns out shorter than what was used in Mester and Bräysy [26] and longer than for Groër et al. [15].

3.7. Impact of algorithmic components

To examine the impact on the performance of CPM of the main algorithmic components, a set of experiments was conducted. In each experiment, the CPM standard was altered to deactivate or remove some components respectively. The experiments are described below.

- Only use Type 1 reinsertion (R1): All TS threads employ Type 1 reinsertion strategy. In this experiment, the other three reinsertion strategies are removed from CPM.
- Only use Type 2 reinsertion (R2): All TS threads employ Type 2 reinsertion strategy.
- Only use Type 3 reinsertion (R3): All TS threads employ Type 3 reinsertion strategy.
- Only use Type 4 reinsertion (R4): All TS threads employ Type 4 reinsertion strategy.
- Only exchange feasible solutions (OF): All TS threads only exchange feasible solutions with the pool. When a TS thread does not improve its starting solution, the solution can still be sent to the solution pool so as to keep the frequency of solution exchange identical.
- No guidance (NG): In this experiment, both intensification and diversification mechanism are deactivated. Set cluster size threshold for diversification to a large number (e.g. 5000) so that the threads for diversification can obtain solutions from any clusters all the time no matter how many solutions a cluster contains. Set the tabu tenures and solution selection rule of the intensification threads identical to those for the diversification threads.

These modified versions were tested on the Golden et al. [24] instances and the average results are compared against those of CPM standard. The comparison is shown in Table 5. At first glance the impact on the performance of the algorithmic components may seem small, but it is in fact crucial because for these well-studied instances, even minute improvements are difficult to obtain. The results thus show that all these algorithmic components contribute to the high performance of CPM.

3.8. Effect of the search effort on performance

To examine the performance of the proposed metaheuristic when dissimilar search effort is employed, the CPM standard was executed with several settings on the Golden et al. [24] instances. In each setting, the total number of iterations is altered. The results are showed in Table 6.

In the table, the first row provides the number of iterations for each experiential setting. The second row presents the average deviations from the best known solutions of the best results for each setting. Likewise, the average deviations of the average results for each setting are shown in the third row. The last row provides the average wall-clock time per instance for each setting. From the results, it is noticeable that the quality of the solutions obtained gradually improves until a peak level is reached as the search effort increases. Additionally, we can see that CPM standard

Table 4
Comparison of results for benchmarks of Li et al. [25].

Instances	Previous best known	Mester and Bräysy [26]	Groër et al. [15] 129p	CPM standard			
				Aver.	Time (min)	SD	Best
21(560)	16 212.74	16 212.74	16 212.83	16 214.12	14.25	1.05	16 212.83
22(600)	14 575.19	14 597.18	14 584.42	14 562.10	19.35	11.98	**14 539.79**
23(640)	18 801.12	18 801.12	18 801.13	18 853.80	18.18	106.69	18 801.13
24(720)	21 389.33	21 389.33	21 389.43	21 390.96	22.41	1.39	21 389.43
25(760)	16 739.84	17 095.27	16 763.72	16 733.07	33.05	16.91	**16 709.44**
26(800)	23 971.74	23 971.74	23 977.73	23 981.30	28.73	1.26	23 980.12
27(840)	17 408.66	17 488.74	17 433.69	17 380.24	38.05	24.26	**17 343.38**
28(880)	26 565.92	26 565.92	26 566.03	26 569.96	33.14	1.88	26 567.23
29(960)	29 154.34	29 160.33	29 154.34	29 157.42	40.85	1.98	29 154.33
30(1040)	31 742.51	31 742.51	31 742.64	31 746.20	51.17	1.60	31 742.64
31(1120)	34 330.84	34 330.84	34 330.94	34 333.66	62.63	2.12	34 330.94
32(1200)	36 919.24	36 928.70	37 185.85	37 188.36	72.36	16.29	37 162.54
Aver. deviation (%)		0.23	0.09	0.07			−0.01
Time (min)		104.30	5.00		36.18		
Runs per instances		1	5	10			10

Table 5
Impact of the algorithmic components.

Experiment	CPM st.	R1	R2	R3	R4	OF	NG
Aver. deviation from BKS (%)	0.11	0.67	0.17	0.19	0.23	0.15	0.15
Average wall-clock time per instance (min)	34.47	33.00	33.87	34.45	35.12	34.63	34.53

Table 6
Comparison of the effect of search effort.

Iterations ($1000 \times \sqrt{\lvert N \rvert}$)	30	60	90	150	180
Aver. deviation from BKS of best results (%)	0.08	0.05	0.01	0.00	0.00
Aver. deviation from BKS of average results (%)	0.18	0.14	0.12	0.11	0.11
Average wall-clock time per instance (min)	6.84	13.87	20.51	34.47	40.95

Table 7
Analyzing the parallel speedup.

Number of processors	8	16	24	72	120	240
Iterations ($1000 \times \sqrt{\lvert N \rvert}$)	450	225	150	50	30	15
Aver. deviation from BKS (%)	0.113	0.109	0.110	0.111	0.116	0.117
Average wall-clock time per instance (min)	101.44	50.88	34.47	11.72	7.00	3.63

can identify solutions at a similar quality to those of Groër et al. [15] and Vidal et al. [23] even when less search effort is utilized.

3.9. *Measuring the parallel speedup*

The parallel speedup is one of the most widely used measures of a parallel algorithm's effectiveness. This metric is defined as the ratio between the sequential and parallel times. The sequential time is the amount of time required for running the algorithm on a single computer, and the parallel time refers to the amount of time required for the parallel computation when using multiple processors. However, in this experiment, we do not compare the parallel time against the sequential time since it may impair the effectiveness of CPM to run it on a single computer. We instead compare the quality of average results and the wall-clock time when using a different number of processors and a fixed amount of search effort (i.e., the total iterations per thread times the number of processors). This experiment is carried out on instances from Golden et al. [24] and the fixed search effort is set to $(((150\,000 \times \sqrt{\lvert N \rvert})\ \text{iterations}) \times (24\ \text{processors}))$. The results are shown in Table 7. From the results, it is observable that, for up to 240 processors, increasing the number of processors generally allows CPM to discover solutions of approximately equivalent quality in roughly linearly reduced time.

4. Conclusions and perspectives

In this paper, we have presented a cooperative parallel metaheuristic for the capacitated vehicle routing problem. The computational experiments on the two sets of large scale CVRP benchmark instances show that the proposed metaheuristic is effective and competitive in comparison to state-of-the-art methods from the literature. Though the instances used have been well-studied, the proposed parallel metaheuristic is still able to identify new best solutions to 10 of the 32 instances within reasonable computational time. From a parallel computation point of view, the proposed parallel metaheuristic is efficient and flexible as it can employ at least up to 240 processors and achieve a roughly linear speedup.

In addition, several new features are introduced in this paper. First, using the structural information (edge residence counters) of solutions enables the solution clustering to be carried out rapidly, even when thousands of solutions are involved. Search history information can thus be extracted from the solution clusters, helping the search to better achieve intensification and diversification. Second, the novel way of implementing reinsertion neighborhoods based on the distance to the depot of customers makes it easier to seek improvement for different parts of the solutions. Moreover, for cooperative search, it appears beneficial to also exchange unfeasible solutions among search threads. The combination of these features largely contributes to the high performance of the proposed metaheuristic.

In this paper, we focused on cooperation and information exchange and used the cooperation concept where all exchanges proceed through the common solution pool. The solution clustering-based intensification and diversification introduced is very general and problem-domain independent. It can be easily applied in different contexts for solving other combinatorial problems. The modification required is to select the solution elements on which common features are built and use them in a similarity measure for the problem in question. Future work will focus on pursuing other approaches of applying the information extracted from the solution clustering and adopt the proposed parallel metaheuristic to other classes of problems.

Acknowledgments

The authors thank Compute Canada and the Norwegian Metacenter for Computational Science (NOTUR) for providing computing resources to conduct the experiments of this research.

References

[1] Le Bouthillier A, Crainic TG. A cooperative parallel metaheuristic for the vehicle routing problem with time windows. Comput Oper Res 2005;32:1685–708.
[2] Crainic TG. Parallel solution methods for vehicle routing problems. In: Golden B, Raghavan S, Wasil E, editors. The vehicle routing problem: latest advances and new challenges. New York: Springer; 2008. p. 171–98.
[3] Alba E, editor. Parallel metaheuristics: a new class of algorithms. Hoboken, NJ: John Wiley & Sons; 2005.
[4] Crainic TG, Toulouse M. Parallel meta-heuristics. In: Gendreau M, Potvin J-Y, editors. Handbook of metaheuristics. 2nd edition.. Springer; 2010. p. 497–541.
[5] Toth P, Vigo D. The vehicle routing problem. SIAM monographs on discrete mathematics and applications. PA: Philadelphia; 2002.
[6] Laporte G. Fifty years of vehicle routing. Transp Sci 2009;43(4):408–16.
[7] Jin J, Crainic TG, Løkketangen A. A parallel multi-neighborhood cooperative tabu search for capacitated vehicle routing problems. Eur J Oper Res 2012;222(3):441–51.
[8] Jin J, Crainic TG, Løkketangen A. A guided cooperative parallel tabu search for the capacitated vehicle routing problem. In: Norsk Informatikkonferanse NIK 2011. Tapir Akademisk Forlag, ISBN 978-82-519-2843-4; 2011. p. 49–60.
[9] Crainic TG, Hail N. Parallel metaheuristics applications. In: Alba E, editor. Parallel metaheuristics: a new class of algorithms. Hoboken, NJ: John Wiley & Sons; 2005. p. 447–94.
[10] Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 2003;35(3):268–308.

[11] Glover F, Laguna M. Tabu search. Kluwer; 1997.
[12] Voß S. Solving quadratic assignment problems using the reverse elimination method. In: Nash S, Sofer A, editors. The impact of emerging technologies on computer science and operations research. Boston: Kluwer; 1995. p. 281–96.
[13] Toth P, Vigo D. The granular tabu search and its application to the vehicle routing problem. INFORMS J Comput 2003;15:333–46.
[14] Yellow PC. A computational modification to the savings method of vehicle scheduling. Oper Res Q 1970;21(2):281–93.
[15] Groër C, Golden B, Wasil E. A parallel algorithm for the vehicle routing problems. INFORMS J Comput 2011;23:315–30.
[16] Savelsbergh MWP. The vehicle routing problem with time windows: minimizing route duration. INFORMS J Comput 1992;4:146–54.
[17] Potvin JY, Rousseau JM. An exchange heuristic for routing problems with time windows. J Oper Res Soc 1995;46:1433–46.
[18] Taillard E, Badeau P, Gendreau M, Geurtin F, Potvin JY. A tabu search heuristic for the vehicle routing problem with soft time windows. Transp Sci 1997;31:170–86.
[19] Flood MM. The traveling-salesman problem. Oper Res 1956;4:61–75.
[20] Kytöjoki J, Nuortio T, Bräysy O, Gendreau M. An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. Comput Oper Res 2007;34:2743–57.
[21] Groër C, Golden B, Wasil E. A library of local search heuristics for the vehicle routing problem. Math Program Comput 2010;2:79–101.
[22] Prins C. A simple and effective evolutionary algorithm for the vehicle routing problem. Comput Oper Res 2004;31(12):1985–2002.
[23] Vidal T, Crainic TG, Gendreau M, Lahrichi N, Rei W. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper Res 2012;60(3):611–24.
[24] Golden BL, Wasil EA, Kelly JP, Chao IM. The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In: Crainic T, Laporte G, editors. Fleet management and logistics. Boston: Kluwer; 1998. p. 33–56.
[25] Li F, Golden BL, Wasil EA. Very large-scale vehicle routing: new test problems, algorithms, and results. Comput Oper Res 2005;32:1165–79.
[26] Mester D, Bräysy O. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. Comput Oper Res 2007;34:2964–75.