Position Papers of the Federated Conference on Computer Science and Information Systems pp. 87–93 DOI: 10.15439/2016F554 ACSIS, Vol. 9. ISSN 2300-5963

Is Your Parallel Algorithm Correct?

Jakub Nalepa
Institute of Informatics
Silesian University of Technology
Akademicka 16
44-100 Gliwice, Poland
Email: jakub.nalepa@polsl.pl

Miroslaw Blocho
Institute of Informatics
Silesian University of Technology
Akademicka 16
44-100 Gliwice, Poland
Email: blochom@gmail.com

Abstract—**Verifying the correctness of parallel algorithms is not trivial, and it is usually omitted in the works from the parallel computation field. In this paper, we discuss in detail how to show that a certain parallel algorithm is correct. This process involves proving its safety and liveness. We perform the in-depth analysis of our parallel guided ejection search (P–GES) for the pickup and delivery problem with time windows, which serves as an excellent case study. P–GES was implemented as a distributed algorithm using the Message Passing Interface library with asynchronous communications, and was validated using the well-known Li and Lim's benchmark containing demanding test instances. We already proved the efficacy of this algorithm and showed that it can retrieve very high-quality (quite often better than the world's best at that time) routing schedules.**

I. Introduction

DESIGNING and implementing parallel algorithms attracted attention of researchers from various fields, including the computational biology, genomics, text processing, pattern recognition, machine learning, optimization, and many others, due to the availability of various parallel architectures. Such approaches allow for solving extremely complex tasks in short time, assuming that: the parallel algorithms are correct and scalable. Proving the correctness of parallel techniques is not trivial (it is much more difficult compared with serial algorithms), and it is omitted in a majority of works belonging to the parallel computation field.

In this paper, we show how to investigate the correctness of a given parallel algorithm. Our parallel guided ejection search technique (P–GES) for minimizing the number of trucks in the NP-hard pickup and delivery problem with time windows, serves as the case study—we analyze its correctness, and show how to accomplish that in a step-by-step manner. In our previous works [1], [2], we experimentally evaluated the Message Passing Interface implementation of P–GES. The extensive experimental study revealed that this algorithm is quite efficient, and it is able to extract very high-quality feasible schedules (often better than the world's best known solutions at that time). The analysis of its correctness presented in this paper therefore complements our previous efforts and theoretically proves that P–GES is correct indeed.

This paper is structured as follows. Section II gives the formulation of the pickup and delivery problem with time windows (PDPTW). Section III reviews the state of the art on solving the PDPTW, and on parallel heuristic algorithms, in order to better contextualize our parallel guided ejection search within the literature. In Section IV, we present the background on verifying the correctness of parallel algorithms, and the correctness of our parallel guided search is proven in Section V. The paper is concluded in Section VI, which also serves as the outlook to our future work.

II. Pickup and Delivery with Time Windows

The PDPTW is a problem of serving a number of transportation requests, each being a pair of the pickup and delivery requests. The PDPTW is therefore defined on a directed graph $G = (V, E)$, with a set V of $C + 1$ vertices. The vertices v_i, $i \in \{1, ..., C\}$, represent the travel points, whereas v_0 denotes the depot (the start and the finish point of each route). A set of edges $E = \{(v_i, v_{i+1}) | v_i, v_{i+1} \in V, v_i \neq v_{i+1}\}$ are the travel connections between each pair of travel points. The travel costs $c_{i,j}$, $i, j \in \{0, 1, ..., C\}$, $i \neq j$, are equal to the distances (in the Euclidean metric) between the travel points. Each request h_i, $i \in \{0, 1, ..., N\}$, where $N = C/2$, is a coupled pair of pickup (P) and delivery (D) customers—these customers are given as p_h and d_h, respectively, where $P \cap D = \emptyset$, and $P \cup D = V \setminus \{v_0\}$ (a customer cannot request both delivery and pickup operations). For each request h_i, the amount of delivered ($q^d(h_i)$) and picked up ($q^p(h_i)$) goods is defined, where $q^d(h_i) = -q^p(h_i)$. Hence, each customer v_i defines its own demand (this is either the delivery or the pick-up demand), service time s_i (note that "serving" the depot does not take time, and $s_0 = 0$), and time window $[e_i, l_i]$ within which the service of this customer should be *started* (however, it can be finished after closing this time slot). Since the fleet is homogenous (let K denote its size), the capacity of each truck is equal (it is given as Q). Each route r, given as $r = \langle v_0, v_1, ..., v_{n+1} \rangle$ in the solution σ (being a set of routes), starts and finishes at the depot, thus $v_0 = v_{n+1}$, and it is an ordered list of visited travel points.

An exemplary PDPTW solution (σ) is rendered in Fig. 1—22 customers (they are divided into the pickup and delivery ones, hence there are 11 pickup-delivery requests) are served in the following three routes: $r_1 = \langle v_0, v_6, v_2, v_1, v_3, v_5, v_4, v_0 \rangle$ (3 requests are handled), $r_2 = \langle v_0, v_8, v_{10}, v_{13}, v_{11}, v_{12}, v_{17}, v_0 \rangle$ (3 requests), $r_3 = \langle v_0, v_{14}, v_{15}, v_{19}, v_{22}, v_{21}, v_{20}, v_{18}, v_{16}, v_9, v_7, v_0 \rangle$ (5 requests). Assuming that (i) the vehicle capacity Q is not exceeded for any vehicle (capacity constraint is satisfied), (ii) the service of every customer starts within its time window (time window constraint), (iii) every customer is served in exactly one route, (iv) every vehicle starts at and returns to the depot within the time window of the depot ($[e_0, l_0]$), and (v) each pickup is performed before the corresponding delivery for each request (precedence constraint), then this solution is feasible.

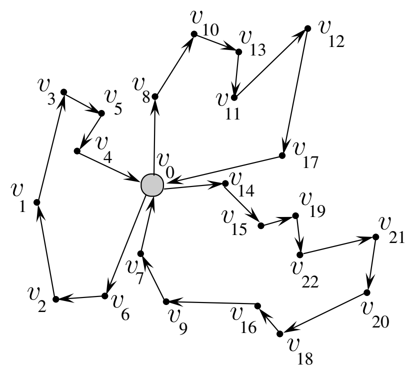


Fig. 1. Exemplary PDPTW solution: 22 clients (11 requests) served in 3 routes.

The PDPTW is a two-objective NP-hard discrete optimization problem. Its primary objective is to minimize the fleet size K, whereas the second objective is to minimize the distance $T = \sum_{i=1}^{K} T_i$, where T_i is the distance traveled in the i-th route. Let σ_A and σ_B denote two feasible PDPTW solutions. σ_A is then of a higher quality compared with σ_B, if $(K(\sigma_A) < K(\sigma_B))$ or $(K(\sigma_A) = K(\sigma_B)$ and $T(\sigma_A) < T(\sigma_B))$. Hence, the solution is of a higher quality if it consists of a lower number of routes, or—if the number of trucks is equal for both solutions—if the total travel distance is shorter.

III. Related Literature

A. Solving the Pickup and Delivery with Time Windows

State-of-the-art algorithms for rich routing problems encompass exact and approximate methods [3]. The former algorithms deliver the exact solutions [4], [5], [6], [7], however they are very difficult to apply in practice, because of their unacceptably large execution times (especially in the case of massively large, real-life problem instances). Also, handling the dynamic changes which are very common in many circumstances (e.g., updating the traffic networks to avoid congestion) are not trivial to incorporate in such algorithms [8]. The exact techniques were discussed in several works [9], [10].

Approximation algorithms include construction and improvement heuristics and metaheuristics [11], [12]. The construction (insertion-based) techniques create solutions from scratch by inserting consecutive requests iteratively into the partial solution [13], [14]. The partial solution encompasses a subset of all transportation requests, therefore is not acceptable and should be expanded to serve other requests (feasibly) as well. On the other hand, improvement heuristics modify an initial solution (very often of a low quality) by applying local search moves (thus, by exploring the neighborhood of this solution) [15], [16]. A number of metaheuristics have been adopted for solving rich VRPs throughout the years, including various tabu searches [4], variable neighborhood searches [17], greedy randomized adaptive search procedures, population-based [18], [19], [20], and agent-based approaches [21], guided ejection searches [22], simulated annealing [16], and more [23].

B. Parallel Heuristic Algorithms

Parallel heuristic algorithms have been explored for solving a bunch of different optimization problems [24], including various VRPs [21], [25]. Co-operative strategies in such parallel heuristic techniques have been discussed and classified in several taxonomies, with the one presented by Crainic et al. being the best established [26], which encompasses three dimensions. The first dimension specifies if the global solving procedure is controlled by a single process (1-control—1C) or by a group of processes (p-control—pC). These processes may co-operate (in co-operative algorithms) or not (if the processing is batched). The second dimension reflects the quantity and quality of the information exchanged between the parallel processes, along with the additional knowledge derived from these exchanges. The four classes are defined for this dimension: Rigid (RS), Knowledge Synchronization (KS), Collegial (C) and Knowledge Collegial (KC). The third dimension concerns the diversity of the initial solutions and search strategies: Same Initial Point / Population, Same Search Strategy (SPSS), Same Initial Point / Population, Different Search Strategies (SPDS), Multiple Initial Points / Populations, Same Search Strategies (MPSS), Multiple Initial Points / Populations, Different Search Strategies (MPDS).

The parallel algorithms were very intensively explored for solving rich routing problems [25], [27], including the PDPTW [1], [2]. The implementations of these algorithms take advantage from the massively-parallel architectures (both with the shared and distributed memory [28]) which are easily accessible nowadays. These techniques deliver extremely high-quality routing schedules in short time, even for enormously large problem instances.

IV. Verification of the Correctness of Parallel Algorithms

Verifying the correctness of a given sequential algorithm encompasses showing that this algorithm: (i) will finish (thus, the termination conditions will be finally met), and (ii) will give a correct result for any correct set of input data. More formally, the correctness may be stated as:

$$\{p\}A\{q\}, \tag{1}$$

where A denotes the algorithm (a set of statements), p is the pre-condition, and q represents the post-condition. Here, the pre-condition specifies which conditions must hold for the input data, and the post-condition reflects what should be satisfied by the results retrieved using the algorithm A. The algorithm is *partially correct* if for any input data satisfying the pre-condition, it gives the correct output data (in accordance with the post-condition) [29] (thus, the input-output relation holds). On the other hand, the algorithm is *totally correct*, if it is partially correct, and—for any input data—it reaches the termination condition (this is not crucial in the case of the partial correctness), and returns the correct output. It is easy to note that proving the total correctness of a sequential algorithm may consist of proving its partial correctness, along with showing that every execution of this algorithm will result in meeting the stopping condition [30].

In the case of parallel algorithms, proving their correctness includes verifying their *safety* and *liveness*. The algorithm is safe, if it can never end up in a forbidden state. To prove that, we need to show that the algorithm is (i) partially correct, (ii) there are no deadlocks (i.e., the processes do not wait for the infinite amount of time for each other to continue the execution), and that (iii) only the processes can safely access the shared resource (*mutual exclusion*). The liveness property of a parallel algorithm is satisfied, if it can be proven that a certain desired condition will eventually happen during the execution of this algorithm [31], [32]. In the case of message-passing techniques—as shown in [30]—it is important to show that the messages are properly sent and received (no matter if the communication is synchronous or asynchronous).

If all of the above-mentioned properties of an analyzed parallel algorithm are proven, then this algorithm is *correct*.

V. Correctness of the Parallel Guided Ejection Search for the PDPTW

The baseline (sequential) version of the GES was proposed in [23], and later enhanced and parallelized in our very recent works [1], [2], [22]. According to the taxonomy mentioned in Section III-B, P–GES is of the pC/C/MPSS type (p-Control, Collegial, Multiple Initial Points, Same Search Strategies).

A. Algorithm Outline

In P–GES, which is an improvement parallel heuristic technique, *p* processes execute in parallel (Algorithm 1, line 1). The initial feasible solution σ contains the number of routes which is equal to the number of transportation requests, hence each request is feasibly served by a separate truck (line 3). Then, the number of serving vehicles in σ is consecutively decreased until the total computation time exceeds the imposed time limit τ_M (lines 5-36), or the desired number of routes has been obtained.

A random route r is removed from σ, and the excluded requests are put into the ejection pool (EP), which stores those transportation requests that have been removed from the schedule—this solution becomes partial (line 7). The penalty counters (indicated as p's), which reflect the difficulty of re-inserting a given request back into the partial solution, for all of the requests are reset (line 8).

If the EP contains unserved transportation requests (lines 9-32), then a single request h_{in} is popped from the EP at the

Algorithm 1 A parallel algorithm to minimize K (P–GES).

```
1: for $P_i \leftarrow P_1$ to $P_p$ do in parallel
2:     $\tau_{last} \leftarrow \tau_{curr}$;
3:     Create an initial solution $\sigma$;
4:     finished ← false;
5:     while not finished do
6:         Save the current feasible solution;
7:         Put requests from a random route $r$ into EP;
8:         Set penalty counters $p[i] \leftarrow 1 (i = 1, 2, \ldots, N)$;
9:         while (EP $\neq \emptyset$) and (not $finished$) do
10:            Select and remove request $h_{in}$ from EP;
11:            if $S_{in}^{fe}(h_{in}, \sigma) \neq \emptyset$ then
12:                $\sigma \leftarrow$ random $\sigma' \in S_{in}^{fe}(h_{in}, \sigma)$;
13:            else
14:                $\sigma \leftarrow$ Squeeze($h_{in}, \sigma$);
15:            end if
16:            if $h_{in}$ is not inserted into $\sigma$ then
17:                $p[h_{in}] \leftarrow p[h_{in}] + 1$;
18:                for $k \leftarrow 1$ to $k_m$ do
19:                    Get $S_{ej}^{fe}(h_{in}, \sigma)$ with min. $\mathcal{P}_{sum}$;
20:                    if $S_{ej}^{fe}(h_{in}, \sigma) \neq \emptyset$ then
21:                        $\sigma \leftarrow$ random $\sigma' \in S_{ej}^{fe}(h_{in}, \sigma)$;
22:                        Add ($h_{out}^{(1)}$, $h_{out}^{(2)}$, ..., $h_{out}^{(k)}$) to EP;
23:                        break;
24:                    end if
25:                end for
26:            end if
27:            $\sigma \leftarrow$ Perturb($\sigma$);
28:            if $\tau_{curr} \geq \tau_{last} + \tau_{coop}$ then
29:                finished← Cooperate($\sigma$);
30:                $\tau_{last} \leftarrow \tau_{curr}$;
31:            end if
32:        end while
33:        if EP $\neq \emptyset$ then
34:            Backtrack to previous feasible solution;
35:        end if
36:    end while
37:    Get the best solution $\sigma_{best}$;
38: end for
```

time (line 10), and it is being re-inserted into the partial solution. If there exist any feasible insertion positions for this request (the set of such positions $S_{in}^{fe}(h_{in}, \sigma)$ is not empty), then a random position is drawn (line 12). If it is not the case, then the request is inserted into σ infeasibly (so that it violates the constraints), and the feasibility of the (possibly partial) solution is being restored in the *squeezing* procedure (line 14). Here, the solution penalty is quantified using the penalty function given as:

$$\mathcal{F}_p(\sigma) = \mathcal{F}_c(\sigma) + \mathcal{F}_{tw}(\sigma), \tag{2}$$

where $\mathcal{F}_c(\sigma)$ and $\mathcal{F}_{tw}(\sigma)$ are the sum of capacity exceeds in σ, and the sum of the time windows violations, respectively. The squeeze function (presented in Algorithm 2) aims at decreasing the value of this function until it reaches zero (thus, the solution is feasible). This is a steepest-descent local search procedure, in which the set $S^{inf}(h_{in}, r, \sigma_t)$ of infeasible solutions is created (considering the insertion of the analyzed transportation request), and the solution with the minimum value of the penalty function is picked up. This process continues until the feasibility is restored, or it is impossible to retrieve a feasible solution (in this case, the solution is backtracked to the initial state).

Algorithm 2 Squeezing an infeasible (possibly partial) solution σ.

```
1: function SQUEEZE($h_{in}$, $\sigma$)
2:     $\sigma_t \leftarrow \sigma' \in S^{inf}(h_{in}, \sigma)$ such that $\mathcal{F}_p(\sigma')$ is minimum;
3:     while ($\mathcal{F}_p(\sigma_t) \neq 0$) do
4:         Randomly choose an infeasible route $r$ in $\sigma_t$;
5:         Find $\sigma'' \in S^{inf}(h_{in}, r, \sigma_t)$ with min. $\mathcal{F}_p(\sigma'')$;
6:         if $\mathcal{F}_p(\sigma'') < \mathcal{F}_p(\sigma_t)$ then
7:             $\sigma_t \leftarrow \sigma''$;
8:         else
9:             break;
10:        end if
11:    end while
12:    if $\mathcal{F}_p(\sigma_t) = 0$ then
13:        return $\sigma_t$;
14:    else
15:        return $\sigma$;
16:    end if
17: end function
```

If the squeeze fails (thus the solution has been backtracked to the previous partial schedule), the penalty counter of the appropriate request ($p[h_{in}]$) is increased (Algorithm 1, line 17), and other requests are ejected from the solution (up to k_m requests; lines 18-25) to insert h_{in} (this request is of a "high priority"). The set $S^{fe}_{ej}(h_{in}, \sigma)$ is formed, and it encompasses the solutions with various combinations of ejected requests (the h_{in} request is inserted to this solution on various positions). Finally, the solution σ'—with the minimum sum of the penalty counters is selected from $S^{fe}_{ej}(h_{in}, \sigma)$ (line 21). Clearly, the ejected requests are pushed to the EP, and should be re-inserted into σ later (line 22). The solution σ is finally perturbed by the local search procedures, in which I feasible (i.e., not violating the constraints) local moves (out-relocate and out-exchange) are executed for the search diversification (line 27). This procedure is visualized in Algorithm 3.

The parallel processes in P–GES co-operate periodically every τ_{coop} seconds (Algorithm 1, line 29) using the asynchronous co-operation scheme. In our previous works [25], [2], we investigated a number of co-operation schemes (they define the co-operation topology, frequency, and the strategies for handling emigrants/immigrants) and showed, that a proper selection of such scheme has a tremendous impact on the algorithm capabilities and behavior.

In P–GES, it is the master process (P_1) which controls the execution time of the algorithm—the signals from P_1

Algorithm 3 Perturbing a feasible (possibly partial) solution σ for the search diversification.

```
1: function PERTURB($\sigma$)
2:     $\sigma_t \leftarrow \sigma$;
3:     for $i \leftarrow 1$ do $I$
4:         Find $\sigma'$ through local search moves on $\sigma_t$;
5:         if $\sigma'$ is feasible then
6:             $\sigma_t \leftarrow \sigma'$;
7:         end if
8:     end for
9:     return $\sigma_t$;
10: end function
```

to either continue or stop the execution are transferred in each co-operation phase. Eventually, all solutions from all processes are gathered in the master, and the best solution σ_{best} is retrieved—this is the final solution delivered by P–GES (line 37).

More details on P–GES can be found in our previous works [1], [2]. These papers include the in-depth analysis of the Message Passing Interface implementation of this algorithm, and discuss the experimental results retrieved for very demanding Li and Lim's benchmark sets (encompassing tests of various sizes and characteristics, e.g., positions of the travel points, and tightness of time windows).

B. Proving the Correctness of P–GES

The input data passed to P–GES include:

- p ($p \geq 1$)—the number of parallel processes. If $p = 1$, then P–GES becomes a sequential algorithm, and its certain components are disabled (e.g., the co-operation between the processes).
- $K_d \geq 0$—the desired number of trucks serving the requests. If $K_d = 0$, then the best feasible solution found using P–GES is returned (i.e., there is no "desired" number of routes, however K should be as minimum as possible).
- τ_{MAX}—the maximum execution time (in seconds) of P–GES.
- τ_{coop}—the co-operation frequency (in seconds).
- k_m ($k_m \geq 1$)—the maximum number of requests that can be ejected from a (possibly partial) solution while inserting a request popped from the EP.
- I ($I \geq 0$)—the number of local search moves applied to perturb a solution.
- Test instance—the definition of the test instance at hand. It specifies the number of transportation requests, the positions of the travel points, their time windows, service times, and demands (either pickup or delivery), and the maximum capacity of trucks. It is worth noting that real-life problems may encompass travel points which are clustered, randomly scattered around the map, or combine both (i.e., there are some customer clusters, but lots of them are random). The problem instances belonging to the Li and Lim's benchmark set perfectly reflect these

scenarios—the exemplary instance structures (with 100 travel points) are visualized in Table I.

The desired solution retrieved using P–GES must satisfy all the constraints discussed in Section II. Therefore, this solution must be *feasible* (otherwise, the routing schedule is incorrect).

As mentioned in Section V-A, P–GES starts with an initial feasible solution (therefore, the constraints are not violated), in which every transportation request is served in a separate route. Then, the attempts to reduce the fleet size are undertaken, until the execution reaches the termination condition (Algorithm 1, line 5)—one random route is analyzed at any time. The current (best) solution is saved (line 6). If removing this route fails, then the partial solution is backtracked to this state (line 34), hence the schedule remains feasible.

Once the ejected customers are pushed into the EP, the solution becomes a partial feasible schedule (no constraints are violated). Then, these transportation requests are put back into the partial solution—first, using the feasible insertion positions (if any). In this case, the feasibility is not violated, and the next request from the EP is popped for insertion. On the other hand, the infeasible solution (in which the request has been re-inserted back infeasibly) is processed with the squeeze procedure. This squeezing retrieves either the feasible solution (if it is possible to restore the correctness of σ using local search moves), or backtracks to the state before this squeezing has been called. In the latter case, other transportation requests are ejected to restore the feasibility of the partial solution, thus it finally becomes feasible. Perturbing a feasible (potentially partial) schedule can deteriorate its quality, however it cannot cause violating the constraints—after calling this procedure, the solution remains feasible. If the EP is empty, then the feasible solution—with the decreased fleet size—becomes the next solution, which is to be processed in the next algorithm iteration. Therefore, the PDPTW solution obtained using P–GES is eventually always feasible.

The co-operation of parallel processes cannot affect the feasibility of the solutions—depending on the co-operation scheme, the receiving process may e.g., replace its own solution with the immigrant (if the immigrant is of a higher quality). Clearly, this operation cannot affect the feasibility of the considered routing schedule. This analysis shows that P–GES is *partially correct*—assuming that the input data are correct, it always retrieves feasible PDPTW solutions.

P–GES may be terminated if either a solution of a desired quality (i.e., with the desired number of routes, K_d) is found, or if the maximum execution time elapsed. In the former case, this solution may be retrieved by the master process (which also controls the execution of other processes in the team, and may send the termination request), or any other process. If the master got this solution, then it sends the termination requests to others (thus, one co-operation phase is enough to stop the parallel algorithm execution). However, if another process ended up with the desired solution, then two co-operation phases would be necessary—first, it sends its best solution to the master, and then the master sends the termination request to other processes. In either case, P–GES finally reaches its stopping condition.

P–GES is a distributed algorithm (there are no shared resources). The co-operation is asynchronous (independently from the selected scheme), and the execution (i.e., optimization of the solution run by a given process) interleaves with the send/receive operations. The order of send/receive operations matter in this case, thus they are executed in an appropriate order depending on the process type (either the master on non-master). Additionally, receiving data is acknowledged by the receiving process during the co-operation (the status of this acknowledgement is periodically checked by the sending process). Since there are no deadlocks and shared resources in P–GES, its safety is proven. The same reasoning may be used to prove the liveness of the algorithm. Since only the master process can force other processes to stop, the situation in which a given process sends to or waits for a message from the process that has already been terminated is not possible. This shows the liveness property of P–GES.

The above investigation revealed that all of the conditions imposed on the parallel algorithms which ensured that the corresponding algorithm is correct are fulfilled by P–GES, for the correct input data (e.g., assuming that the test instance at hand is solvable). Therefore, P–GES is a correct parallel algorithm.

□

VI. Conclusions and Outlook

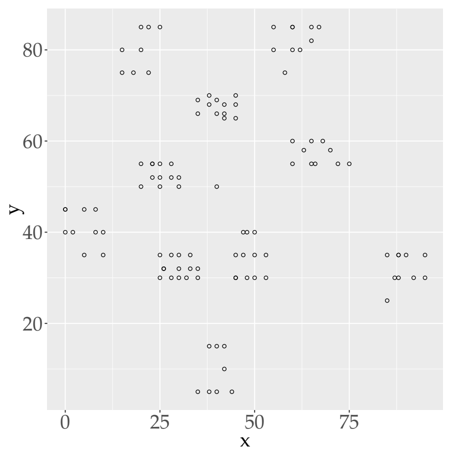
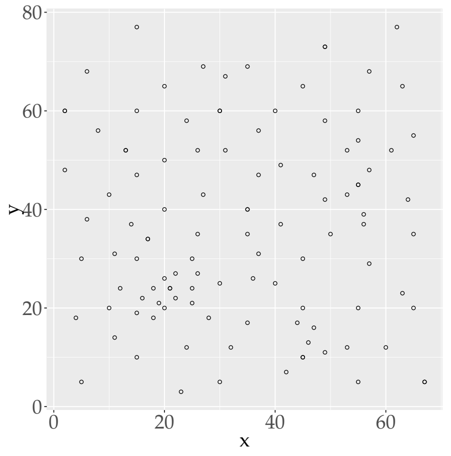
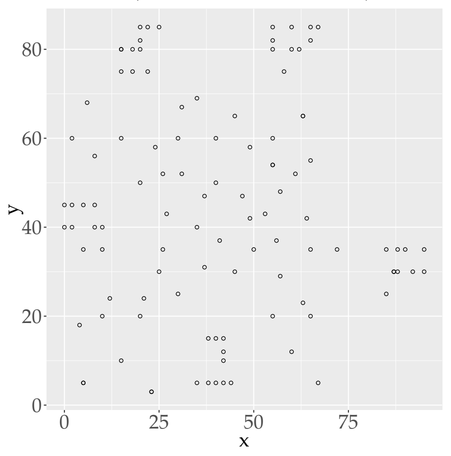
In this paper, we analyzed the correctness of our parallel guided ejection search algorithm for solving the PDPTW. We proved that the algorithm is correct—this involved showing its liveness and safety. This investigation served as an extensive case study for showing how to prove the correctness of parallel algorithms. This approach may be easily tailored for proving the correctness of other parallel algorithms, especially those tackling complex (however not only transportation) discrete optimization problems.

Our current research is focused on implementing a parallel memetic algorithm (a hybrid of a genetic algorithm and some local refinement procedures) for minimizing the travel distance in the PDPTW. Memetic algorithms were proven extremely efficient in solving a wide range of optimization and pattern recognition problems [33], [34], [35], [36], [37], [38]. Then, we will work on a parallel version of this algorithm (we already proved that our parallel memetic approach for the VRP with time windows is correct [30]). Combining the parallel guided ejection search discussed in this paper with the parallel memetic algorithm will enable us to create a full optimization framework for solving rich routing problems [39], especially the PDPTW.

Acknowledgment

This research was supported by the National Science Centre under research Grant No. DEC-2013/09/N/ST6/03461, and performed using the infrastructure supported by the POIG.02.03.01-24-099/13 grant: “GeCONiI—Upper Silesian

TABLE I
EXEMPLARY LI AND LIM'S INSTANCE STRUCTURES COMPOSED OF CLUSTERED, RANDOMIZED, AND MIXED CUSTOMERS (FOR 100 TRAVEL POINTS, BEING EITHER THE PICKUP OR DELIVERY CUSTOMERS).

Clustered	Random	Mixed (both clustered and random)
		

Center for Computational Science and Engineering", and the Intel CPU and Xeon Phi platforms provided by the MICLAB project No. POIG.02.03.00.24-093/13.

REFERENCES

[1] M. Blocho and J. Nalepa, "A parallel algorithm for minimizing the fleet size in the pickup and delivery problem with time windows," in *Proc. of 22nd European MPI Users' Group Meeting*, ser. EuroMPI '15. New York, USA: ACM, 2015, pp. 15:1–15:2. [Online]. Available: http://doi.acm.org/10.1145/2802658.2802673

[2] J. Nalepa and M. Blocho, "A parallel algorithm with the search space partition for the pickup and delivery with time windows," in *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015, Krakow, Poland, November 4-6, 2015*, 2015, pp. 92–99. [Online]. Available: http://dx.doi.org/10.1109/3PGCIC.2015.12

[3] L. Grandinetti, F. Guerriero, F. Pezzella, and O. Pisacane, "The multi-objective multi-vehicle pickup and delivery problem with time windows," *Social and Beh. Sc.*, vol. 111, pp. 203 – 212, 2014.

[4] W. P. Nanry and J. W. Barnes, "Solving the pickup and delivery problem with time windows using reactive tabu search," *Transportation Research*, vol. 34, no. 2, pp. 107 – 121, 2000.

[5] J.-F. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," *Oper. Res.*, vol. 54, no. 3, pp. 573–586, 2006. [Online]. Available: http://dx.doi.org/10.1287/opre.1060.0283

[6] R. Baldacci, E. Bartolini, and A. Mingozzi, "An exact algorithm for the pickup and delivery problem with time windows," *Operations Research*, vol. 59, no. 2, pp. 414–426, 2011. [Online]. Available: http://dx.doi.org/10.1287/opre.1100.0881

[7] A. Bettinelli, A. Ceselli, and G. Righini, "A branch-and-price algorithm for the multi-depot heterogeneous-fleet pickup and delivery problem with soft time windows," *Mathematical Programming Computation*, vol. 6, no. 2, pp. 171–197, 2014. [Online]. Available: http://dx.doi.org/10.1007/s12532-014-0064-0

[8] B. Bernay, S. Deleplanque, and A. Quilliot, "Routing on dynamic networks: GRASP versus genetic," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, September 7-10, 2014.*, 2014, pp. 487–492. [Online]. Available: http://dx.doi.org/10.15439/2014F52

[9] J.-F. Cordeau, G. Laporte, and S. Ropke, *The Vehicle Routing Problem: Latest Advances and New Challenges*. Boston, MA: Springer, 2008, ch. Recent Models and Algorithms for One-to-One Pickup and Delivery Problems, pp. 327–357. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-77778-8_15

[10] R. Baldacci, A. Mingozzi, and R. Roberti, "Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints," *European Journal of Operational Research*, vol. 218, no. 1, pp. 1 – 6, 2012.

[11] H. Akeb, A. Bouchakhchoukha, and M. Hifi, "A beam search based algorithm for the capacitated vehicle routing problem with time windows," in *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems, Kraków, Poland, September 8-11, 2013.*, 2013, pp. 329–336. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6644021

[12] ——, *Recent Advances in Computational Optimization: Results of the Workshop on Computational Optimization WCO 2013, FedCSIS 2013*. Cham: Springer International Publishing, 2015, ch. A Three-Stage Heuristic for the Capacitated Vehicle Routing Problem with Time Windows, pp. 1–19. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-12631-9_1

[13] Q. Lu and M. M. Dessouky, "A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows," *European Journal of Operational Research*, vol. 175, no. 2, pp. 672 – 687, 2006.

[14] C. Zhou, Y. Tan, L. Liao, and Y. Liu, "Solving the multi-vehicle pick-up and delivery problem with time widows by new construction heuristic," in *Proc. IEEE CISDA*, vol. 2, 2006, pp. 1035–1042. [Online]. Available: http://dx.doi.org/10.1109/ISDA.2006.253754

[15] H. Li and A. Lim, "A metaheuristic for the pickup and delivery problem with time windows," in *Proc. IEEE ICTAI*, 2001, pp. 160–167. [Online]. Available: http://dx.doi.org/10.1109/ICTAI.2001.974461

[16] S. N. Parragh, K. F. Doerner, and R. F. Hartl, "A survey on pickup and delivery problems," *Journal fur Betriebswirtschaft*, vol. 58, no. 1, pp. 21–51, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11301-008-0033-7

[17] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation Science*, vol. 40, no. 4, pp. 455–472, 2006. [Online]. Available: http://dx.doi.org/10.1287/trsc.1050.0135

[18] G. Pankratz, "A grouping genetic algorithm for the pickup and delivery problem with time windows," *OR Spectrum*, vol. 27, no. 1, pp. 21–41, 2005. [Online]. Available: http://dx.doi.org/10.1007/s00291-004-0173-7

[19] Y. Nagata and S. Kobayashi, *Proc. PPSN XI*. Heidelberg: Springer, 2010, ch. A Memetic Algorithm for the Pickup and Delivery Problem with Time Windows Using Selective Route Exchange Crossover, pp. 536–545. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15844-5_54

[20] M. Cherkesly, G. Desaulniers, and G. Laporte, "A population-based metaheuristic for the pickup and delivery problem with time windows and LIFO loading," *Computers & Operations Research*, vol. 62, pp. 23 – 35, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054815000829

[21] P. Kalina and J. Vokrínek, "Parallel solver for vehicle routing and pickup and delivery problems with time windows based on agent negotiation," in *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2012, pp. 1558–1563. [Online]. Available: http://dx.doi.org/10.1109/ICSMC.2012.6377958

[22] J. Nalepa and M. Blocho, *Intelligent Information and Database Systems: Proc. 8th Asian Conference, ACIIDS 2016*. Heidelberg: Springer, 2016, ch. Enhanced Guided Ejection Search for the Pickup and Delivery Problem with Time Windows, pp. 388–398. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-49381-6_37

[23] Y. Nagata and S. Kobayashi, "Guided ejection search for the pickup and delivery problem with time windows," in *Proc. EvoCOP*, ser. LNCS. Springer, 2010, vol. 6022, pp. 202–213. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12139-5_18

[24] T. G. Crainic and M. Toulouse, "Parallel meta-heuristics," in *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science, M. Gendreau and J.-Y. Potvin, Eds. Springer US, 2010, vol. 146, pp. 497–541. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-1665-5_17

[25] J. Nalepa and M. Blocho, "Co-operation in the parallel memetic algorithm," *International Journal of Parallel Programming*, vol. 43, no. 5, pp. 812–839, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10766-014-0343-4

[26] T. G. Crainic and H. Nourredine, "Parallel meta-heuristics applications," in *Parallel Metaheuristics: A New Class of Algorithms*, M. Gendreau and J.-Y. Potvin, Eds. Wiley, 2005, pp. 447–494. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-1665-5_17

[27] G. Senarclens de Grancy and M. Reimann, "Evaluating two new heuristics for constructing customer clusters in a vrptw with multiple service workers," *Central European Journal of Operations Research*, vol. 23, no. 2, pp. 479–500, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10100-014-0373-4

[28] R. Banos, J. Ortega, C. Gil, F. de Toro, and M. G. Montoya, "Analysis of OpenMP and MPI implementations of meta-heuristics for vehicle routing problems," *Applied Soft Computing*, vol. 43, pp. 262 – 275, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494616300862

[29] Z. Manna, "Mathematical theory of partial correctness," *Journal of Computer and System Sciences*, vol. 5, no. 3, pp. 239 – 253, 1971. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022000071800351

[30] M. Blocho, "A parallel memetic algorithm for the vehicle routing problem with time windows," Ph.D. dissertation, Silesian University of Technology, 2013, (in Polish).

[31] S. Owicki and L. Lamport, "Proving liveness properties of concurrent programs," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 455–495, 1982. [Online]. Available: http://doi.acm.org/10.1145/357172.357178

[32] Z. Czech, *Introduction to Parallel Computing*. PWN, 2013.

[33] S. Wrona and M. Pawelczyk, "Controllability-oriented placement of actuators for active noise-vibration control of rectangular plates using a memetic algorithm," *Archives of Acoustics*, vol. 38, no. 4, pp. 529–536, 2013. [Online]. Available: http://dx.doi.org/10.2478/aoa-2013-0062

[34] J. Nalepa and M. Kawulok, "A memetic algorithm to select training data for support vector machines," in *Proc. of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '14. New York, USA: ACM, 2014, pp. 573–580. [Online]. Available: http://doi.acm.org/10.1145/2576768.2598370

[35] K. Siminski, *Man–Machine Interactions 4: 4th International Conference on Man–Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, October 6–9, 2015*. Cham: Springer International Publishing, 2016, ch. Memetic Neuro-Fuzzy System with Big-Bang-Big-Crunch Optimisation, pp. 583–592. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23437-3_50

[36] J. Nalepa, M. Cwiek, and M. Kawulok, "Adaptive memetic algorithm for the job shop scheduling problem," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/IJCNN.2015.7280409

[37] J. Nalepa and M. Blocho, "Adaptive memetic algorithm for minimizing distance in the vehicle routing problem with time windows," *Soft Computing*, vol. 20, no. 6, pp. 2309–2327, 2016. [Online]. Available: http://dx.doi.org/10.1007/s00500-015-1642-4

[38] J. Nalepa and M. Kawulok, "Adaptive memetic algorithm enhanced with data geometry analysis to select training data for SVMs," *Neurocomputing*, vol. 185, pp. 113 – 132, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231215019839

[39] M. Cwiek, J. Nalepa, and M. Dublanski, *Intelligent Information and Database Systems: Proc. 8th Asian Conference, ACIIDS 2016*. Heidelberg: Springer, 2016, ch. How to Generate Benchmarks for Rich Routing Problems?, pp. 399–409. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-49381-6_38