2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing

A Parallel Memetic Algorithm for the Pickup and Delivery Problem with Time Windows

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

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

***Abstract*—Solving the pickup and delivery problem with time windows (PDPTW) is a vital research topic due to its NP-hardness and its numerous practical applications. In this paper, we propose an island-model parallel memetic algorithm for minimizing the distance in the PDPTW. In this algorithm, the processes execute the same memetic algorithm and co-operate to guide the optimization efficiently. An extensive experimental study revealed that the MPI implementation of the proposed approach retrieves very high-quality routing schedules. The analysis is coupled with appropriate statistical tests.**

***Keywords*-Parallel algorithm, memetic algorithm, PDPTW.**

I. Introduction

Modeling various real-life transportation problems has been attracting the research attention over last decades [1]. There are numerous formulations of rich routing problems which reflect different scenarios. The maximum load capacities appear in the capacitated vehicle routing problem (CVRP) [2], whereas the time windows, in which the customers must be visited are given in the VRP with time windows [3]. In the pickup and delivery problem with time windows (PDPTW), the transportation requests (pickup or delivery) are to be served [4]. A pickup location must be visited before the delivery location (precedence constraint), the capacity of trucks cannot be exceeded (capacity constraint), and the service of each customer must be started within its time window (time window constraint) [5].

Applying exact algorithms for solving rich routing problems is still a challenging task due to their high time complexity. Hence, approximate methods, which do not ensure obtaining exact solutions but work in acceptable time, are of research interest. Since the PDPTW is a two-objective problem (the main objective is to minimize the fleet size, and the secondary one is to optimize the distance), such heuristics and metaheuristics may be designed independently for both optimization stages. The algorithms to minimize the fleet size encompass simulated annealing, neighborhood searches, genetic algorithms, and other metaheuristics [6], [7]. Techniques for minimizing the distance include hybrid algorithms [8], [9], memetic algorithms [5], particle swarm optimization [10], and more [11], [12].

A. Contribution

In this work, we introduce the island-model parallel memetic algorithm (abbreviated as P–LCS-SREX) for minimizing the total distance in the PDPTW. In P–LCS-SREX, the parallel processes co-operate periodically and exchange the best solutions found to date, in order to guide the search efficiently. It was shown that the co-operation topology, frequency and approaches for handling immigrant/emmigrant solutions drastically affect the search capabilities of the parallel algorithms [13]. Here, we exploit the ring co-operation scheme, which was proved to provide high exploration abilities of such distributed co-operative techniques [13]. Each process in P–LCS-SREX runs the memetic algorithm (MA) based on the improved selective route exchange crossover (SREX), originally proposed in [5], and very recently substantially improved [14]. In this MA, a low-quality population evolves to optimize the travel distance.

P–LCS-SREX was implemented using the Message Passing Interface (MPI) library. This implementation is thoroughly examined experimentally on the Li and Lim's benchmark tests. We present an in-depth analysis of the results carried out to investigate the algorithm behavior and convergence capabilities—experiments were executed on an SMP cluster. We show that P–LCS-SREX is able to retrieve significantly higher-quality solutions compared with its sequential counterpart. Also, this study revealed that the sequential algorithm converges to low-quality routing schedules very fast—these solutions cannot be further improved. This problem is resolved in P–LCS-SREX, which exposes very good exploration capabilities. To verify the significance of the results, we performed the Friedman tests.

B. Paper structure

Section II describes the PDPTW. In Section III, we review the state of the art concerning solving the PDPTW, and enumerate the parallel algorithms applied for tackling rich routing problems. The proposed parallel algorithm is discussed in Section IV. Section V contains the in-depth analysis of the experimental results coupled with the statistical tests. Section VI concludes the paper.

2377-5750/17 $31.00 © 2017 IEEE
DOI 10.1109/PDP.2017.75

II. Problem Formulation

The PDPTW is a problem of serving transportation requests, being the pairs of pickup and delivery operations. This problem can be modeled on a directed graph $G = (V, E)$, with a set V of $C+1$ vertices (customers, where v_0 is the depot). A set of edges $E = \{(v_i, v_{i+1}) | v_i, v_{i+1} \in V, v_i \neq v_{i+1}\}$ (travel connections). The travel costs $c_{i,j}$, $i, j \in \{0, 1, ..., C\}$, $i \neq j$, are the distances between the travel points (Euclidean metric). Each request h_i, $i \in \{0, 1, ..., N\}$, where $N = C/2$, is a pair of pickup (P) and delivery (D) customers—p_h and d_h, where $P \cap D = \emptyset$, and $P \cup D = V \setminus \{v_0\}$. For each h_i, the amount of delivered ($q^d(h_i)$) and picked up ($q^p(h_i)$) goods is known, and $q^d(h_i) = -q^p(h_i)$. Each customer v_i defines its demand, service time s_i ($s_0 = 0$), and time window $[e_i, l_i]$ within which its service must be started (it can be finished after it has been closed). Since the fleet is homogenous (with K denoting its size), the capacity of each truck is equal and denoted as Q. Each route $r = \langle v_0, v_1, ..., v_{n+1} \rangle$ in the solution σ (set of routes), starts and finishes at $v_0 = v_{n+1}$.

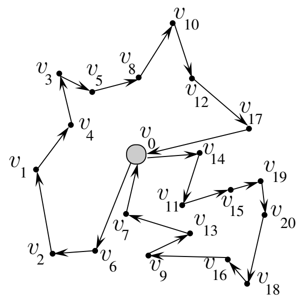


Figure 1. An example PDPTW solution with 20 clients served in 2 routes.

An example PDPTW solution (σ) is shown in Fig. 1—20 customers are served in $K = 2$ routes: $r_1 = \langle v_0, v_6, v_2, v_1, v_4, v_3, v_5, v_8, v_{10}, v_{12}, v_{17}, v_0 \rangle$ and $r_2 = \langle v_0, v_{14}, v_{11}, v_{15}, v_{19}, v_{20}, v_{18}, v_{16}, v_9, v_{13}, v_7, v_0 \rangle$. This solution is feasible if: (i) Q is not exceeded for any truck, (ii) the service of every customer starts within its time window, (iii) every customer is served in one route, (iv) every vehicle starts and returns to v_0 within its time window, (v) every pickup is done before the corresponding delivery.

The primary objective of the PDPTW is to minimize the fleet size (K). The second objective is to minimize the travel distance $T = \sum_{i=1}^{K} T_i$, where T_i is the distance of the *i-th* route. Let σ_A and σ_B denote two feasible PDPTW solutions. σ_A is of higher quality than σ_B, if ($K(\sigma_A) < K(\sigma_B)$) or ($K(\sigma_A) = K(\sigma_B)$ and $T(\sigma_A) < T(\sigma_B)$).

III. Related Literature

State-of-the-art techniques for the PDPTW include exact and approximate approaches (which do not ensure delivering exact solutions, but work much faster). The exact algorithms are still applied for solving quite small problem instances, due to their unacceptable computation time. Such algorithms include dynamic programming based techniques, column generation methods, branch-and-cut and branch-and-price solvers, hybrid algorithms, and many more [15]. A single objective of the PDPTW (either minimizing K or T) is often tackled in the exact approaches.

Approximate algorithms for minimizing K in the PDPTW encompass construction and improvement heuristic techniques—the former are aimed at creating a feasible solution by inserting the unserved requests into a partial solution, whereas the latter ones improve the quality of an initial solution by executing optimization moves and exploring the solution neighborhood [6]. Thanks to the hierarchical objective of the PDPTW, the algorithms for both optimization stages are usually designed independently. Hence, it is possible to tailor the algorithms for both objectives. The route-minimization algorithms include tabu and neighborhood searches [5], [16], [17], population-based approaches [18], particle swarm optimizations, and other [6]. Applying memetic algorithms—the hybrids of genetic approaches and local-search procedures exploiting the knowledge attained during the evolution or extracted beforehand—has been studied for the PDPTW [5], [14], and other routing problems [3], [13], [19]. They well balance the exploration and exploitation of the solution space, and retrieve high-quality schedules [20].

The main goal of the parallel heuristic algorithms is to solve large problem instances in reasonable time [21]. Such approaches were applied to a plethora of optimization tasks, including VRPs. These algorithms encompass parallel evolutionary algorithms [13], parallel simulated annealing, tabu searches [22], agent-based techniques [4], and many others. The co-operation of processes in parallel co-operative techniques is crucial, because it can help guide the search efficiently towards high-quality solutions faster (but may also cause the premature convergence of the search) [13].

Parallel heuristic approaches can be categorized using several taxonomies [23], reflecting three dimensions. The first dimension specifies if the solving procedure is controlled by one process (1-control—1C) or many (p-control—pC) processes. The second dimension addresses the quantity and quality of the information being exchanged between processes, as well as the additional knowledge derived from these exchanges—four classes of approaches are highlighted for this dimension: Rigid (RS), Knowledge Synchronization (KS), Collegial (C) and Knowledge Collegial (KC). The third dimension considers the diversity of the initial solutions (across the parallel processes) 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).

IV. Parallel algorithm

In this section, we present the proposed parallel memetic algorithm (P–LCS-SREX) to minimize the PDPTW distance. According to the previously discussed taxonomy, P–LCS-SREX is of the pC/C/MPSS type. Its sequential version was presented in [5], and recently improved in [14].

A. Algorithm outline

P–LCS-SREX aims at minimizing the travel distance in the PDPTW. In this work, we exploit the parallel guided ejection search [24] to minimize the fleet size (Algorithm 1, line 1), and then to generate the initial solutions for each process (also referred to as the *island*) (lines 3–5). This parallel route-minimization heuristics can be conveniently replaced by another (perhaps more efficient) algorithm.

Algorithm 1 Parallel memetic algorithm for the PDPTW.

```
1: $\sigma_1 \leftarrow$ **PARALLEL-ROUTE-MINIMIZATION**();
2: $m \leftarrow$ routes count of $\sigma_1$;
3: **parfor** $P_i \leftarrow P_1$ **to** $P_p$ **do**
4:     Generate the population of $N_{pop}$ feasible solutions;
5: **end parfor**
6: **parfor** $P_i \leftarrow P_1$ **to** $P_p$ **do**
7:     *done* $\leftarrow$ **false**;
8:     **while not** *done* **do**
9:         Determine $N_{pop}$ random pairs $(\sigma_A^p, \sigma_B^p)$;
10:        **for** all pairs $(\sigma_A^p, \sigma_B^p)$ **do**
11:            **if** $\sigma_A^p = \sigma_B^p$ **then**
12:                $\sigma_A^p \leftarrow$ **PERTURB**($\sigma_A^p$);
13:            **end if**
14:            $\sigma_{best}^c \leftarrow \sigma_A^p$;
15:            $\{\sigma_1^c, \sigma_2^c, \dots, \sigma_{N_{ch}}^c\} \leftarrow$ execute $N_{ch}$ times
               **GENERATE-CHILD-SOLUTION**($\sigma_A^p, \sigma_B^p$);
16:            **for** $i \leftarrow 1$ **to** $N_{ch}$ **do**
17:                $\sigma_i^c \leftarrow$ **EDUCATE**($\sigma_i^c$);
18:                **if** $T(\sigma_i^c) < T(\sigma_{best}^c)$ **then**
19:                    $\sigma_{best}^c \leftarrow \sigma_i^c$;
20:                **end if**
21:            **end for**
22:            $\sigma_A^p \leftarrow \sigma_{best}^c$;
23:        **end for**
24:        Co-operate (ring co-operation scheme);
25:        *done* $\leftarrow$ **CHECK-STOPPING-CONDITION**
26:    **end while**
27: **end parfor**
28: **return** best solution $\sigma_{best}$ across all populations;
```

P–LCS-SREX includes p components executed as processes $P_1, P_2, \dots, P_p$ (Algorithm 1, line 6). Each island evolves N_{pop} feasible solutions to minimize T (lines 8–26). First, the pairs of solutions are selected for crossover (line 9)—here, we use the AB-selection, in which every solution from the population is selected as the first parent, and as the second parent exactly once. If it appears that the parents have the same structure, one solution is perturbed. This operation involves applying a number of local search moves (they are rendered in Fig. 2 and Fig. 3), which consecutively affect the structure of the feasible solution.

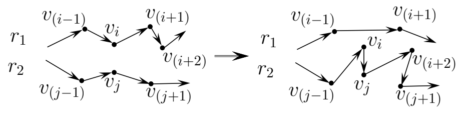


Figure 2. Out–relocate operator applied to two routes r_1 and r_2 from a solution σ. This figure was inspired by [24].

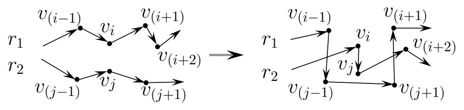


Figure 3. Out–exchange operator applied to two routes r_1 and r_2 from a solution σ. This figure was inspired by [24].

As presented in Algorithm 2, the quality of the solution may be deteriorated, but it remains feasible (the moves which can lead to violating the feasibility of a solution are not allowed). The perturb procedure helps diversify the search and avoid crossing over very similar solutions (the quality of the child solution would most likely not be boosted in this case). Then, N_{ch} children are created for each pair of parents (for more details on the SREX and LCS-SREX crossover operators, along with the illustrative examples, see [5], [14]). Finally, each child is educated (Algorithm 1, line 17). It resembles the perturb operation (see Algorithm 3), but does *not* allow for applying the moves which lower the quality of the solution (line 5) (hence, only improvement and feasible local search moves are possible). The best offspring replaces the first child for all pairs (Algorithm 1, line 22)—it is the post-selection scheme.

Algorithm 2 Perturbing a feasible solution σ.

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

The processes in P–LCS-SREX co-operate every g_{coop} generations (line 24) using the asynchronous scheme discussed at length in Section IV-B. Only the master process

Algorithm 3 Educating a feasible solution σ.

```
 1: function EDUCATE($\sigma$)
 2:     $\sigma_t \leftarrow \sigma$;
 3:     for $i \leftarrow 1$ do $I$
 4:         Apply local search moves to $\sigma_t$ to generate $\sigma'$;
 5:         if $\sigma'$ is feasible and $T(\sigma') < T(\sigma)$ then
 6:             $\sigma_t \leftarrow \sigma'$;
 7:         end if
 8:     end for
 9:     return $\sigma_t$;
10: end function
```

(P_1) controls the execution time—the signals from P_1 to either continue or to stop the computations are transferred in each co-operation phase. It is worth mentioning, that P–LCS-SREX may be terminated if (i) the maximum execution time has elapsed, (ii) the solution of desired quality has been retrieved, or (iii) the maximum number of generations has been exceeded. Finally, the best solution σ_{best} (with the smallest T across all islands) is determined (line 28).

B. Co-operation between processes

Selecting a co-operation scheme has a tremendous impact on the co-operative parallel algorithms. In P–LCS-SREX, we exploit the ring co-operation topology which proved to be able to guide the search very efficiently and to deliver best asymptotic results for various classes of rich routing problems [13]. In this topology, every co-operation is triggered by the master process, which sends its best solution to the next process in the ring (P_2). It compares the received solution with its best one, decides whether to append it to the current population, and sends the best solution to next process P_3. The received solution is appended to the population only if has not been added to the population already. The last process in the pipeline (P_p) retrieves the best solution from all of the processes, and sends it to the master. Hence, the master process receives the best solution found so far. It is easy to note, that each process P_i possesses at least as good solution as during the previous co-operation round. This co-operation chain is visualized in Fig. 4—the master process (which initializes the co-operation) is rendered in light red, and the arrows show the direction of the data flow during the co-operation phase.

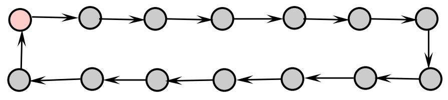


Figure 4. The ring co-operation topology. The master process is rendered in light red, and the arrows show the data flow direction.

The co-operation between the processes is carried out for enhancing the quality of the feasible routing schedules. Here, the best solutions propagate through the co-operation chain of parallel processes. Additionally, this procedure should help escape the local minima into which some of the processes can converge during the optimization (this would not be possible in the batch mode). This may be reflected in longer convergence times of the co-operative algorithms compared with their serial counterparts, which find lower-quality solutions faster, and get stuck in the local minima.

V. Experimental results

A. The MPI implementation

P–LCS-SREX was implemented in `C++` using the Message Passing Interface (MPI). The processes communicate according to the co-operation topology described in Section IV-B. In a two-step data-passing procedure, only the T is sent to a neighbor process at first, along with the information whether a complete solution will be sent in the second step (i.e., if the travel distance has changed since the last co-operation). The complete solution data is sent only if this happened, and the best solution has been improved since the last co-operation phase. We therefore avoid sending solutions of the already-known quality (they would not effectively guide the search and could result in an unnecessary overhead). The entire co-operation process is asynchronous—it means that the communication is interleaved with the processing of each island.

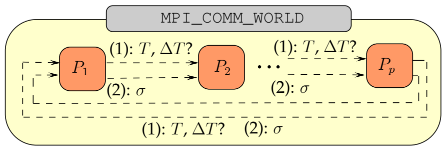


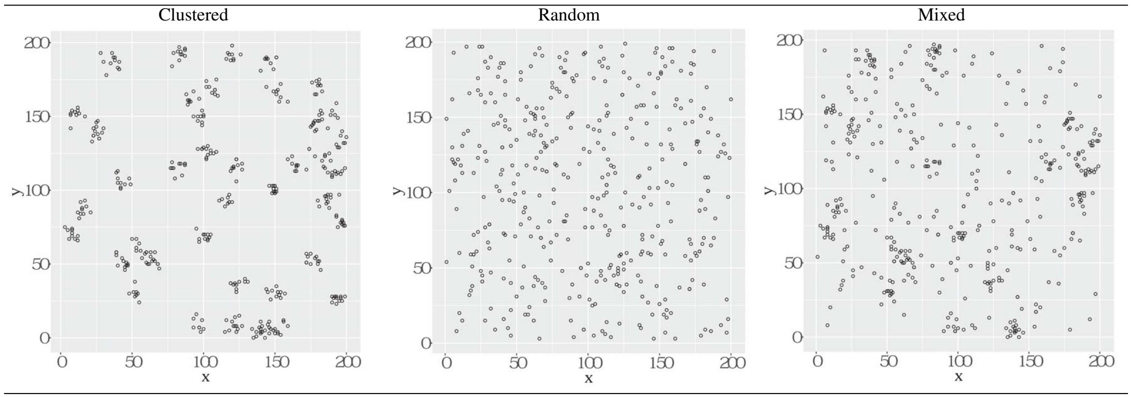
Figure 5. The MPI implementation of the communication. The dotted arrows indicate the two-phase data flow through the co-operation chain.

The parallel processes are grouped into the default communicator `MPI_COMM_WORLD` (Fig. 5—the two-step data-passing procedure is shown as dotted arrows). The process P_i receives data from P_{i-1} and sends data to P_{i+1}. It is carried out with the same argument list in each co-operation phase. To optimize the communication, the persistent requests (`MPI_Request`) were created for all operations:

- `ISendNoReq` (request for sending K),
- `IRecvNoReq` (request for receiving K),
- `ISendSolReq` (request to send the full solution),
- `ISendSolReq` (request to receive the full solution).

Therefore, the number of MPI requests that need to be created equals to a quadrupled number of the processes in a given algorithm run. The list of communication arguments is bound to the persistent requests only once by executing `MPI_Send_Init` or `MPI_Recv_Init` functions. To initiate the communication between the processes P_i and P_j, the function `MPI_Start(req)` is called, where `req`

Table I
THE CUSTOMER LOCATIONS IN THE 400-CUSTOMER BENCHMARK TESTS OF VARIOUS LOCATION CHARACTERISTICS, CONTAINING CLUSTERED (THE C1 AND C2 LI AND LIM'S CLASSES), RANDOMIZED (R1 AND R2), AND MIXED (RC1 AND RC2) CUSTOMERS.



corresponds to the particular MPI request for a given pair of processes. At the end of the algorithm execution, all persistent requests are freed by calling `MPI_Request_Free`.

All `send` and `receive` operations are performed asynchronously. For this reason, it is necessary to wait for the sent requests to be finished—the `MPI_Test` function makes it possible to check whether a given request already completed or not, so that the waiting is handled asynchronously.

B. Settings

The computations were carried out on the supercomputer with Intel Xeon Quad Core (2.33 GHz) nodes with 12 MB level 3 cache. The nodes were connected by the Infiniband DDR fat-free network (throughput 20 Gbps, delay 5 μs). The cluster was executing Linux operating system, and the code was compiled using Intel 10.1 compiler and MVAPICH1 v. 0.9.9 MPI library. The maximum execution time for P–LCS-SREX was set to $\tau = 10$ minutes, and the co-operation was executed every 5 generations (note that the time between the consecutive co-operation phases will be different for various problem instances). The experiments were performed for $p = \{1, 16, 32, 48, 64, 80\}$ parallel islands. The number of perturbing moves was $I = 50$. P–LCS-SREX was executed 5 times for each Li and Lim's benchmark test instance (we analyzed ten 400-customer problem instances belonging to different classes—see Section V-C), for each p.

We exploited the parallel guided ejection search [24] to (i) minimize K, and to (ii) generate the initial populations which later undergo the evolution. The maximum time of optimizing the fleet size was bounded by $\tau_K = 60$ seconds, whereas the maximum time of generating the population (for each island) was set to $\tau_G = 900$ seconds. The populations consisted of $N_{pop} = 10$ (and $N_{ch} = 20$) individuals (its size is relatively small to verify if the parallel algorithm is able to successfully cope with the potential *diversity crisis*).

Table II
INVESTIGATED LI AND LIM'S BENCHMARK INSTANCES.

Instance	K_{WB}	T_{WB}	K
lc1_4_1	40	7152.06	43
lc1_4_2	38	8007.79	41
lr1_4_1	40	10639.75	40
lr1_4_2	31	9968.19	33
lr2_4_1	8	9726.88	11
lr2_4_2	7	9440.93	10
lrc1_4_1	36	9124.52	38
lrc1_4_2	31	8346.06	33
lrc2_4_1	12	7454.14	14
lrc2_4_2	10	7424.72	13
Average	25.3	8728.50	27.6

C. Li and Lim's benchmark tests

P–LCS-SREX was tested on Li and Lim's[1] benchmark tests. Six classes of tests (c1, c2, r1, r2, rc1 and rc2) reflect various real-life factors (e.g., maximum truck capacities or the tightness of time windows). The c1 and c2 classes include clustered customers, while in r1 and r2 the locations are random. The rc1 and rc2 classes contain a mix of random and clustered customers (the examples are visualized in Table I). The classes c1, r1 and rc1 have smaller capacities and tighter time windows compared with c2, r2 and rc2 (a smaller number of trucks should be necessary to handle requests in c2, r2 and rc2). Instances have unique names, lα_β_γ, where α is the class, β relates to the number of customers (4 for 400), and γ is the identifier ($\gamma = 1, 2, \ldots, 10$).

We focus on 400-customer tests belonging to several classes (as summarized in Table II, along with the world's best results—annotated with the WB subscript). Since P–LCS-SREX minimizes the T values, we generate the populations of solutions with K routes (K is very close to

[1]For instance definitions and world's best results see: http://www.sintef.no/projectweb/top/pdptw/li--lim-benchmark/, ref. date: December 1, 2016.

Table III
THE RESULTS (THE FINAL T AND THE INITIAL T_{init} VALUES, AND THE NUMBER OF GENERATIONS g) OBTAINED USING P–LCS-SREX (THE RESULTS ARE AVERAGED ACROSS ALL P–LCS-SREX EXECUTIONS FOR ALL INVESTIGATED PROBLEM INSTANCES). THE BEST AVERAGE RESULTS ARE BOLDFACED, WHEREAS THE BACKGROUND OF THE CELL WITH THE WORST AVERAGE RESULTS IS GRAYED.

	Class→	c1	r1	r2	rc1	rc2	Average
	Min. T_{init}	10231.69	14039.91	16756.37	11342.98	13331.53	13140.49
	Min. T	7303.67	10588.31	8847.07	8659.17	6608.75	8401.39
	Min. g	504.92	995.5	138.5	732	194.5	513.08
	Avg. T_{init}	10254.84	14214.51	17075.26	11521.27	13716.52	13356.48
$p = 1$	Avg. T	7320.50	10895.99	8885.79	8673.12	6684.86	8492.05
	Avg. g	659.05	1007.77	152.08	779.29	218.21	563.28
	Max. T_{init}	10485.57	14438.88	17529.62	11691.92	14152.25	13659.65
	Max T	7616.76	11088.63	8912.04	8694.15	6751.52	8612.62
	Max g	959	1027.5	165	836	265	650.5
	Min. T_{init}	10239.35	14230.35	16189.38	11427.89	13528.71	13123.13
	Min. T	7221.70	10311.59	8796.33	8499.77	6542.08	8274.29
	Min. g	112	121	34	115	50.5	86.5
	Avg. T_{init}	10455.23	14339.15	17137.41	11577.63	14209.69	13543.82
$p = 16$	Avg. T	7221.96	10397.57	8811.04	8513.46	6565.49	8301.90
	Avg. g	115.55	124.9	39	120	53.7	90.63
	Max. T_{init}	10572.64	14563.64	17859.42	11719	15110.72	13965.08
	Max T	7222.51	10528.80	8826.49	8550.74	6607.46	8347.20
	Max g	119.50	130.50	42.50	123.5	58	94.8
	Min. T_{init}	10267.04	13886.95	16633.10	11237.66	13824.77	13169.90
	Min. T	7221.70	10238.64	8809.36	8475.06	6549.97	8258.94
	Min. g	110	119	34.50	110.50	49	84.6
	Avg. T_{init}	10521.34	14195.79	17069.36	11474.45	14168.46	13485.88
$p = 32$	Avg. T	7222.52	10326.02	8825.69	8522.99	6555.83	8290.61
	Avg. g	113	123.20	35.50	118.60	52.90	88.64
	Max. T_{init}	10733.46	14510.52	17266.23	11746.10	14495.13	13750.29
	Max T	7225.65	10465.09	8864.32	8644.43	6564.90	8352.88
	Max g	115.50	127	37	126	56	92.30
	Min. T_{init}	10315.37	14138.05	16878.50	11330.50	13514.55	13235.39
	Min. T	7221.84	10227.97	8805.79	8487.16	6539.88	8256.53
	Min. g	105.50	122	34	112	44	83.50
	Avg. T_{init}	10553.94	14463.69	17169.29	11586.92	14201.46	13595.06
$p = 48$	Avg. T	7263.33	10292.77	8822.53	8503.61	6548.01	8286.05
	Avg. g	117.60	128.60	35.10	117.20	48.65	89.43
	Max. T_{init}	10765.78	14715.66	17429.64	11758.68	14834.72	13900.89
	Max T	7300.96	10395.89	8839.22	8537.35	6556.10	8325.90
	Max g	154	140	36	121.50	55.50	101.40
	Min. T_{init}	10143.13	14184.69	16355.31	11065.05	13228.83	12995.40
	Min. T	7221.70	10233.09	8815.02	8455.54	6539.94	8253.06
	Min. g	105.50	122	32	111	43	82.70
	Avg. T_{init}	10475.75	14396.17	16638.19	11422.70	13986.03	13383.77
$p = 64$	Avg. T	7225.83	10302.77	8827.47	8490.39	6558.60	8281.01
	Avg. g	128.05	126.02	33.40	116.20	49.20	90.57
	Max. T_{init}	10786.91	14516.93	16839.01	11650.44	14685.76	13695.81
	Max T	7235.36	10388.69	8846.71	8544.06	6597.03	8322.37
	Max g	166.50	129.50	34.50	120.50	56	101.40
	Min. T_{init}	10068.08	14278.92	16540.46	11286.78	13516.34	13138.12
	Min. T	7221.70	10184.93	8807.26	8444.44	6544.19	**8240.50**
	Min. g	104	119	30	110.50	44.50	81.60
	Avg. T_{init}	10355.40	14466.24	17081.26	11573.02	13934.80	13482.14
$p = 80$	Avg. T	7223.81	10318.20	8824.04	8476.39	6555.84	**8279.66**
	Avg. g	108.70	124.60	32.72	115.90	47	85.78
	Max. T_{init}	10471.68	14622.13	17575.98	11830.69	14438.80	13787.85
	Max T	7231.98	10535.36	8848.14	8555.10	6576.32	**8321.38**
	Max g	112	130	36.50	120.50	50	89.80

the world's best K_{WB})—such solutions could be retrieved within the imposed maximum execution time of the route minimization algorithm (we do not include the c2 instances in this analysis, since the parallel guided ejection search failed to elaborate solutions with K's close to K_{WB} within τ_K). The set of world's best results contains schedules obtained using both sequential and parallel algorithms.

D. Analysis and discussion

In this section, we summarize the experimental results elaborated using the proposed parallel memetic algorithm. Table III contains the results obtained for all p values (the results are averaged across all algorithm executions, and all problem instances for the investigated Li and Lim's benchmark classes)—the background of the cells with the worst T's are grayed, whereas the best T values across all p's are boldfaced. The results show that increasing the number of parallel islands helps boost the quality of the final solutions—the best schedules are retrieved using P–LCS-SREX executed on $p = 80$ parallel islands. P–LCS-SREX allowed for obtaining the feasible solutions with the T values which are significantly smaller compared with the currently best-known results (see Table III and Table II). For $p = 80$ processes, P–LCS-SREX gave T values decreased by 5.24% on average in this case. The number of generations analyzed during the P–LCS-SREX execution is significantly smaller compared with the serial memetic algorithm. Although the co-operation phase is very fast (and linear with respect to the number of parallel islands), it induces the additional parallel overhead during the execution. Importantly, the large numbers of generations processed by the serial algorithm indicate that the search process got stuck in the local minima—if it happens, then executing a single generation is extremely fast, because there are no feasible local search moves which can robustly improve the quality of the offspring solutions in the education procedure (hence, the neighborhood of the current population is narrowed and well-exploited). It does not occur for $p > 1$—the distributed populations explore different parts of the solution space, and its promising parts are exploited (i) by propagating the best-fitted solutions during the co-operation, and (ii) in the local-search refinements of the child routing schedules.

The convergence time (after which the best solutions in the population could not be further improved) is rendered in Fig. 6. This plot proves the above-mentioned observation—the serial algorithm converges to relatively low-quality solutions very fast, and it cannot escape the local minima afterwards. Increasing p's allows for obtaining better-fitted individuals (as shown in Table III), as well as for exploring larger parts of the solution space. This is reflected in longer convergence times, showing that the exploration of unknown solutions coupled with the intensive exploitation of the best individuals in the populations (of all islands) can lead to better asymptotic routing schedules. Also, the convergence time of P–LCS-SREX for $p = 80$ islands was quite close to the execution time limit for e.g., r2 test instances (the problems with larger truck capacities and longer scheduling horizons appeared more challenging), therefore increasing this limit would most likely allow for retrieving even higher-quality PDPTW solutions. This would not be possible for $p = 1$, since the optimization process got stuck in the local minima which could not be later escaped (also, the minimum and maximum convergence times are close to the average time for most classes). This problem could be addressed by re-generating the populations which ended up in the diversity crisis (i.e., copying a smaller number of the best individuals and creating the others randomly to keep N_{pop} constant). This re-generation would require deciding if the crisis occurred, which is not a trivial task [25].

To investigate the statistical significance of the results, we performed the Friedman tests (here, we consider the T values averaged across all instances, and all P–LCS-SREX runs). Also, it is often difficult to assess the performance of different algorithms across different datasets (or problem instances)—the Algorithm A may outperform the Algorithm B for the first dataset, and can give notably

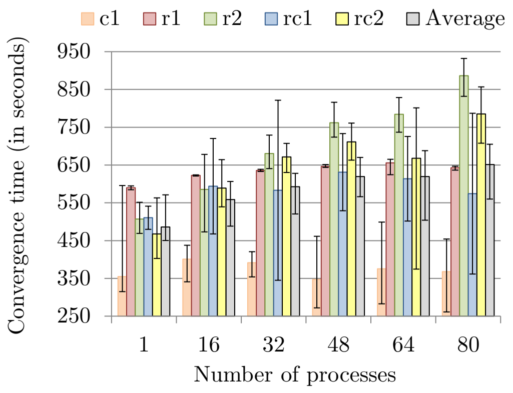


Figure 6. Average convergence time (in seconds) of all P–LCS-SREX variants (the bars show the minimum and the maximum convergence times).

worse results for the second dataset. In Table IV, we gather the results of the Friedman ranking test (at p-value equal to 0.05). These results reveal that the parallel algorithm retrieved statistically better routing schedules compared with its sequential counterpart. Although increasing the number of parallel islands not necessarily leads to elaborating higher-quality solutions on average (the ranking values appear similar for $p \geq 48$), the best results (across all algorithm executions) are significantly better for larger p's (see Table III). Also, P–LCS-SREX executed on $p = 80$ islands gave the best routing schedules in the worst-case scenario.

Table IV
THE RESULTS OF THE FRIEDMAN TEST ($p = 0.05$).

$p\downarrow$	Ranking	Different from
1	6.00	$P = 16, 32, 48, 64, 80$
16	3.20	$P = 1$
32	3.40	$P = 1$
48	2.40	$P = 1$
64	3.40	$P = 1$
80	2.60	$P = 1$

VI. CONCLUSION AND OUTLOOK

In this paper, we proposed a parallel memetic algorithm (P–LCS-SREX) to minimize the travel distance in the PDPTW. In P–LCS-SREX, the parallel processes co-operate periodically in an asynchronous co-operation scheme, and exchange the best solutions found up to date. The experimental study performed to investigate the efficacy of the MPI implementation of P–LCS-SREX using selected problem instances from the famous Li and Lim's benchmark set revealed that our approach retrieves very high-quality solutions, and outperforms the serial algorithm when the travel distance is to be minimized. The statistical tests showed that these differences are important. Also, we showed that the exploration of the solution space is well-balanced with the exploitation of the best individuals in the parallel populations—the sequential algorithm converged fast to low-quality schedules which were not further improved. This problem was mitigated in P–LCS-SREX—the convergence time was slightly increased, and the quality of final solutions was notably boosted with the use of parallelism.

Our ongoing work includes designing an adaptive version of P–LCS-SREX, in which its parameters are dynamically adjusted on the fly. P–LCS-SREX might be potentially deployed on Hadoop utilizing the MapReduce paradigm. We work on the load balancing in P–LCS-SREX, based on the assessment of the regions of the search space which should be intensively explored. We aim at comparing the performance of P–LCS-SREX with other parallel algorithms to solve the PDPTW, as well as other routing problems, since incorporating new constraints into P–LCS-SREX should be straight-forward. Also, we work on more efficient algorithms for minimizing the number of routes. Finally, we plan to analyze the speedup and efficiency of P–LCS-SREX (this is not trivial, and these standard notions should be slightly modified in P–LCS-SREX, as indicated in [24]).

Acknowledgment

This research was supported by the National Science Centre under research Grant No. DEC-2013/09/N/ST6/03461. We thank the Academic Computer Centre in Gdańsk (TASK CI), where the computations of our project were carried out.

References

[1] Z. J. Czech and P. Czarnas, "Parallel simulated annealing for the vehicle routing problem with time windows," in *Proc. PDP*, 2002, pp. 376–383.

[2] Y. Niu, S. Wang, J. He, and J. Xiao, "A novel membrane algorithm for capacitated vehicle routing problem," *Soft Comput.*, vol. 19, no. 2, pp. 471–482, 2015.

[3] J. Nalepa and M. Blocho, "Adaptive memetic algorithm for minimizing distance in the vehicle routing problem with time windows," *Soft Comput.*, vol. 20, no. 6, pp. 2309–2327, 2016.

[4] P. Kalina and J. Vokrinek, "Parallel solver for vehicle routing and pickup and delivery problems with time windows based on agent negotiation," in *IEEE SMC*, 2012, pp. 1558–1563.

[5] Y. Nagata and S. Kobayashi, "Guided ejection search for the pickup and delivery problem with time windows," in *Proc. of EvoCOP*, ser. LNCS. Springer Berlin Heidelberg, 2010, vol. 6022, pp. 202–213.

[6] 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.

[7] F. A. Santos, G. R. Mateus, and A. S. Da Cunha, "The pickup and delivery problem with cross-docking," *Comput. Oper. Res.*, vol. 40, no. 4, pp. 1085–1093, 2013.

[8] R. Kammarti, S. Hammadi, P. Borne, and M. Ksouri, "Improved tabu search in an hybrid evolutionary approach for the pickup and delivery problem with time windows," in *Proc. ITS*, Sept 2005, pp. 148–153.

[9] R. Bent and P. V. Hentenryck, "A two-stage hybrid algorithm for pickup and delivery vehicle routing with time windows," *Comput. Oper. Res.*, vol. 33, no. 4, pp. 875–893, 2006.

[10] P. Sombuntham and V. Kachitvichyanukul, "Multi-depot vehicle routing problem with pickup and delivery requests," *AIP Conference Proceedings*, vol. 1285, no. 1, 2010.

[11] M. I. Hosny and C. L. Mumford, "Constructing initial solutions for the multiple vehicle pickup and delivery problem with time windows," *J. of King Saud Univ.*, vol. 24, no. 1, pp. 59 – 69, 2012.

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

[13] J. Nalepa and M. Blocho, "Co-operation in the parallel memetic algorithm," *Int. J. of Parallel Programming*, vol. 43, no. 5, pp. 812–839, 2015.

[14] M. Blocho and J. Nalepa, "LCS-based selective route exchange crossover for pickup and delivery with time windows," (Working paper).

[15] 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," *Math. Prog. Comp.*, vol. 6, no. 2, pp. 171–197, 2014.

[16] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transp. Sc.*, vol. 40, no. 4, pp. 455–472, 2006.

[17] J. Nalepa and M. Blocho, "Enhanced guided ejection search for the pickup and delivery problem with time windows," in *Intelligent Information and Database Systems: 8th Asian Conference, ACIIDS 2016, Proc., Part I*, N. T. Nguyen, B. Trawiński, H. Fujita, and T.-P. Hong, Eds. Springer, 2016, pp. 388–398.

[18] M. Cherkesly, G. Desaulniers, and G. Laporte, "A population-based metaheuristic for the pickup and delivery problem with time windows and LIFO loading," *Comp. & Op. Research*, vol. 62, pp. 23 – 35, 2015.

[19] Y. Nagata, O. Bräysy, and W. Dullaert, "A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows," *Comp. & Op. Res.*, vol. 37, no. 4, pp. 724–737, 2010.

[20] M. Blocho and J. Nalepa, "A parallel algorithm for minimizing the fleet size in the pickup and delivery problem with time windows," in *Proc. EuroMPI*. New York, NY, USA: ACM, 2015, pp. 15:1–15:2.

[21] T. G. Crainic and M. Toulouse, "Parallel meta-heuristics," in *Handbook of Metaheuristics*, ser. Int. Series in Op. Res. & Manag. Sc., M. Gendreau and J.-Y. Potvin, Eds. Springer US, 2010, vol. 146, pp. 497–541.

[22] S. Jagiello and D. Zelazny, "Solving multi-criteria vehicle routing problem by parallel tabu search on GPU," *Procedia Computer Science*, vol. 18, no. 0, pp. 2529 – 2532, 2013.

[23] T. G. Crainic and H. Nourredine, "Parallel meta-heuristics applications," in *Parallel Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Wiley, 2005, pp. 447–494.

[24] 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.

[25] J. Nalepa and M. Kawulok, "A memetic algorithm to select training data for support vector machines," in *Proc. of the 2014 Annual Conf. on Genetic and Evol. Comput., GECCO 2014*, ser. GECCO '14. USA: ACM, 2014, pp. 573–580.