Two Phased Hybrid Local Search for the Periodic Capacitated Arc Routing Problem

Yuning Chen [a] and Jin-Kao Hao [b,c,*]

[a] *College of Information System and Management, National University of Defense Technology, Changsha 410073, PR China*

[b] *LERIA, Université d'Angers, 2 bd Lavoisier, 49045 Angers Cedex 01, France*

[c] *Institut Universitaire de France, 1 rue Descartes, 75231 Paris Cedex 05, France*

European Journal of Operational Research
https://doi.org/10.1016/j.ejor.2017.06.025

Abstract

The periodic capacitated arc routing problem (PCARP) is a challenging general model with important applications. The PCARP has two hierarchical optimization objectives: a primary objective of minimizing the number of vehicles (F_v) and a secondary objective of minimizing the total cost (F_c). In this paper, we propose an effective two phased hybrid local search (HLS) algorithm for the PCARP. The first phase makes a particular effort to optimize the primary objective while the second phase seeks to further optimize both objectives by using the resulting number of vehicles of the first phase as an upper bound to prune the search space. For both phases, combined local search heuristics are devised to ensure an effective exploration of the search space. Experimental results on 63 benchmark instances demonstrate that HLS performs remarkably well both in terms of computational efficiency and solution quality. In particular, HLS discovers 44 improved best known values (new upper bounds) for the total cost objective F_c while attaining *all* the known optimal values regarding the objective of the number of vehicles F_v. To our knowledge, this is the first PCARP algorithm reaching such a performance. Key components of HLS are analyzed to better understand their contributions to the overall performance.

Keywords: Heuristics; Capacitated arc routing; Bi-level optimization; Constrained combinatorial search.

* Corresponding author.
Email addresses: cynnudt@hotmail.com (Yuning Chen), jin-kao.hao@univ-angers.fr (Jin-Kao Hao).

1 Introduction

Due to their theoretical hardness and practical importance, the periodic capacitated arc routing problem (PCARP) as well as many of its closely related problems in logistics have attracted considerable research effort in the last decades [8]. Compared to the popular capacitated arc routing problem (CARP) [15], the PCARP requires that the tasks are served for a certain number of times over a given multi-period horizon. The PCARP is typically encountered in waste collection applications, where we want to design a plan to collect the daily waste on each street in the city. In the PCARP that was first introduced in [20], streets may require several services for a multi-period time horizon (e.g., one week) according to a service pattern (e.g., a street requiring two services can be serviced by a Monday-Thursday or Tuesday-Friday pattern). The PCARP is to schedule vehicles to cover the required services of each day over the time horizon while optimizing two hierarchical objectives: a *primary* objective of minimizing the number of vehicles used over the time horizon (F_v) and a *secondary* objective of minimizing the total cost (F_c).

The PCARP is computationally challenging since it generalizes the classical and NP-hard CARP [15]. Compared to its single-period special case–CARP–which has been intensively studied in the last decades (e.g., [1,3,17,25,27,30,31]), the PCARP is somewhat less investigated. Due to its intrinsic intractability, existing research on the PCARP focuses mainly on designing effective heuristics to find high-quality near-optimal solutions in a reasonable time frame. As a first attempt to solve this problem, Chu *et al.* [5] presented several constructive heuristics. Later, two advanced heuristic algorithms were proposed: the Memetic Algorithm (LMA) by Lacomme *et al.* [22] and the Scatter Search algorithm (SS) by Chu *et al.* [6]. Both approaches adapted the representation scheme and search operators of the classical CARP to the PCARP, and applied a greedy heuristic to build elite initial solutions of the population. Kansou and Yassine [19] introduced an Ant Colony heuristic with an efficient constructive procedure which outperformed the previous methods on a set of instances. Finally, Mei *et al.* [28] presented another memetic algorithm with route-merging (MARM) which clearly dominated all previous PCARP approaches, making a significant improvement in PCARP solving. This approach will serve as the main reference for our algorithm assessment.

For a comprehensive literature review, we mention a close relative of the PCARP, called the Periodic Vehicle Routing Problem (PVRP), which is the vertex routing counterpart of the PCARP. The PVRP appeared earlier than the PCARP and consequently has received more research attention (e.g., [7, 9, 11, 12]). Different from the PCARP investigated in this work which is a bi-level optimization problem, the PVRP involves a single objective (the total cost objective). Moreover, the PVRP can be considered to be inherently less complex than the PCARP if we compare their single-period special cases (CARP vs. VRP). Indeed, a CARP with n tasks corresponds to a VRP with $2n + 1$ vertices [23]. These observations also confirm

the challenge of solving the PCARP compared to the PVRP.

In this work, we propose a two phased hybrid local search (HLS) approach for solving the PCARP with the following motivations and contributions.

- We notice that to handle the two hierarchical objectives, the existing studies on the PCARP typically optimize an aggregated weight function which is a linear combination of the two hierarchical objectives. Even if this approach is simple to implement, it does not explicitly recognize the priority of the primary objective and the algorithms using this approach need to explore a very large search space including many irrelevant solutions. Contrary to this objective aggregation approach, our HLS algorithm proposed in this paper relies on: 1) a first phase which focuses on the minimization of the number of vehicles, and 2) a second phase which uses the resulting number of vehicles as an upper bound to strongly constrain the optimization process.
- The proposed HLS algorithm integrates dedicated search operators and heuristics to ensure an effective search of both phases. To obtain an initial PCARP solution with a small number of vehicles, the first phase of HLS employs a specific tabu search procedure to *evenly* assign the tasks among the different periods of the time horizon and applies a heuristic CARP algorithm to generate vehicle routes for each period. In order to further ameliorate this initial PCARP solution, the second phase of HLS relies on two complementary local search procedures to reduce both the number of vehicles and the total cost. In particular, HLS uses the number of vehicles of the initial solution (from the first phase) as an upper bound to discard all candidate solutions whose number of vehicles is larger than the upper bound, and thus only explores a largely reduced search space.
- We assess our HLS algorithm on three sets of 63 popular benchmark instances in the literature. Our computational results indicate that HLS competes very favorably with the current best PCARP algorithms and is able to reach all the known optimal values in terms of F_v, and discovers 44 improved best values (new upper bounds) in terms of F_c which can be used to evaluate new PCARP algorithms. To our knowledge, no previous algorithm achieves such a performance.

The reminder of the paper is organized as follows. We introduce the PCARP in Section 2 and then present the HLS algorithm in Section 3. We show a performance assessment of the proposed algorithm and an analysis of the key elements of HLS in Sections 4 and 5 respectively, followed by concluding remarks in Section 6.

2 Problem Description and Solution Representation

2.1 Problem description

Given a m-period time horizon $H = \{1, 2, ..., m\}$ and an undirected graph $G(V, E)$ with vertex set V and edge set E, a set of required edges (also called tasks hereafter) E_R ($E_R \subset E$) and a fleet of identical vehicles with a capacity of Q that are based at the depot vertex v_d ($v_d \in V$). Let n be the number of required edges (i.e., $n = |E_R|$). Each edge $e = (i, j) \in E_R$ (a task), which is considered as a pair of arcs $< i, j >$ and $< j, i >$, is associated with a traversal cost ($tc(e)$). Additionally, each task $t \in E_R$ is associated with a service cost $sc(t)$, a service frequency $f(t)$ (based on which, an allowable service pattern set $ASP(t)$ is also given) and a demand vector $d(t) = \{d_1(t), d_2(t), ..., d_m(t)\}$ where $d_x(t)$ ($x = 1, ..., m$) indicates the intraperiod demand of period x of task t.

Let $ad(t, p, h)$ denote the accumulated demand of task t ($t \in E_R$) in period $h \in H$, where task t is served by pattern p ($p \in ASP(t)$). We recall that a pattern depicts the number of services provided for a task over a time horizon, and the periods when the service is performed. Once the pattern for a task is determined, its accumulated demand in a particular service period is calculated by summing up all intraperiod demands between last service period and the current one. For instance, a pattern $p_0 = \{2, 5\}$ is selected for task t_0 indicating t_0 is serviced on the second day (i.e., Tuesday) and fifth day (i.e., Friday) of the week, then the accumulated demand of task t_0 on Tuesday is $ad(t_0, p_0, 2) = d_5(t_0) + d_6(t_0) + d_7(t_0) + d_1(t_0)$ and on Friday is $ad(t_0, p_0, 5) = d_2(t_0) + d_3(t_0) + d_4(t_0)$.

The PCARP amounts to deciding a pattern p ($p \in ASP(t)$) for each task t ($t \in E_R$) and to designing a set of vehicle routes for each period h ($h \in H$), with the purpose of minimizing the number of vehicles (F_v) used over the time horizon H as the primary objective, and minimizing the total cost of all vehicle routes (F_c) as the second objective, while respecting the following constraints: 1) each vehicle route starts and ends at the depot v_d; 2) each task t ($t \in E_R$) is served no more than once in each period h ($h \in H$); 3) the service pattern selected for each task t ($t \in E_R$) must be from its allowable service pattern set $ASP(t)$; 4) the total demand serviced on the route of a vehicle must not exceed the vehicle capacity Q. A mathematical formulation of the PCARPA was described in [28], which is based on a solution representation different from the representation we adopted (described below).

2.2 Solution representation

Our HLS algorithm uses the following solution representation to encode the candidate solutions of the PCARP. First, each task (i.e., each required edge) is assigned

two IDs $(a, a+n)$ $(a = 1, ..., n)$ to represent the two associated arcs of the task. We also define a dummy task with 0 as its task ID and both its head and tail vertices being the depot vertex v_d. This dummy task is to be inserted somewhere in the solution as a route delimiter. A candidate solution S of the PCARP is then represented by m (number of periods) CARP solutions, i.e., $S = \{S_1, S_2, ..., S_m\}$. Suppose each CARP solution S_i $(i \in \{1, ..., m\})$ involves n_i tasks and r_i vehicle routes, S_i can then be encoded as an order list of $(n_i + r_i + 1)$ task IDs among which $(r_i + 1)$ are dummy tasks: $S_i = \{S_i(1), S_i(2), ..., S_i(n_i + r_i + 1)\}$, where $S_i(j)$ is a task ID (an arc of the task or a dummy task) in the j^{th} position of S_i. S_i can also be written as a set of r_i routes: $S_i = \{0, R_{i1}, 0, R_{i2}, 0, ..., 0, R_{ir_i}, 0\}$, where R_{ij} denotes the j^{th} route composed of $|R_{ij}|$ task IDs (arcs), i.e., $R_{ij} = \{R_{ij}(1), R_{ij}(2), ..., R_{ij}(|R_{ij}|)\}$, with $R_{ij}(k)$ being the task ID at the k^{th} position of R_{ij}. Let $dist(u, v)$ denote the shortest path distance between the head vertex of arc u ($head(u)$) and the tail vertex of arc v ($tail(v)$). The primary objective value $F_v(S)$ and the secondary objective value $F_c(S)$ of the candidate solution S can be expressed as

$$F_v(S) = max_{i \in H}(r_i) \tag{1}$$

$$F_c(S) = \sum_{i=1}^{m} \sum_{j=1}^{n_i+r_i} (tc(S_i(j)) + dist(S_i(j), S_i(j+1))) \tag{2}$$

The total load $load(R_{ij})$ of a route R_{ij} $(j \in \{1, ..., r_i\}, i \in H)$ is given by

$$load(R_{ij}) = \sum_{k=1}^{|R_{ij}|} ad(R_{ij}(k), sp(R_{ij}(k)), i) \tag{3}$$

where $sp(R_{ij}(k))$ is the pattern selected for task $R_{ij}(k)$.

Notice that Expression (1) is not the primary objective, but indicates how the primary objective value $F_v(S)$ of a candidate solution $S = \{S_1, S_2, ...S_m\}$ is calculated, which is the maximum number of vehicles used by the different routes over m periods of H in the solution.

3 Main Scheme of HLS and Algorithm Components

The HLS algorithm (Algorithm 1) starts with its first phase to generate an initial feasible PCARP solution satisfying the constraints given at the end of Section 2.1. This is achieved by the $InitSol$ procedure (Section 3.1), which determines a balanced assignment of the tasks over the time horizon (with a dedicated tabu search procedure) and designs a set of routes for each period (with an existing route building procedure). The number of vehicles used in the initial solution naturally constitutes an upper bound of the first objective F_v, denoted as $nvUB$. To further improve the input solution and avoid unpromising solutions, the second phase of HLS (Section 3.2) examines only feasible solutions and discards any candidate solution with a

F_v value higher than $nvUB$. Specifically, the second phase applies two local search procedures to improve both the pattern assignment and vehicle routes, and triggers a perturbation procedure (i.e., accepting some controlled deteriorating solutions) when a local optimum trap is encountered. The HLS algorithm continues until the best solution cannot be improved for W consecutive rounds.

Algorithm 1: Main Scheme of HLS for the PCARP

Data: P: a $PCARP$ instance; W: max allowed number of consecutive non-improving rounds
Result: the best solution S^* found

{First phase - Generate an initial solution with a small number of vehicles} ;
$S \leftarrow InitSol()$;
$S^* \leftarrow S$; /* S^* records the global best solution */
$nvUB \leftarrow F_v(S)$;
{Second phase - Improve the solution by using nvUB as an upper bound} ;
while $noImp < W$ **do**
    $S_c \leftarrow S^*$;
    while *Improving solutions can be found* **do**
        $(S, S^*) \leftarrow route\_improve(S, nvUB)$;
        $(S, S^*) \leftarrow pattern\_improve(S, nvUB)$;
    $(S, S^*) \leftarrow pattern\_perturb(S, nvUB)$;
    if S^* *improves on* S_c **then**
        $noImp \leftarrow 0$
    else
        $noImp \leftarrow noImp + 1$
return S^*

3.1 First phase: generate an initial solution to bound the number of vehicles

Among the two objectives of the PCARP, the first objective (minimizing the used vehicles) has a higher priority over the second objective (minimizing the total cost). Thus, a solution with a smaller F_v value is always better than a solution with a larger F_v value, regardless of their total costs. Consequently, given a feasible solution S_0 with $F_v(S_0)$ vehicles, it is useless to examine any solution S with $F_v(S) > F_v(S_0)$ since S is necessarily worse than S_0. Based on this consideration, we first generate an initial solution with a F_v value as small as possible in order to effectively prune the search. For this purpose, a natural idea is to *evenly* allocate the tasks of E_R to each period, leading to the following demand balancing problem (DBP).

$$max \ \min_{i \in H}\Big(\sum_{t \in E_R} \sum_{p \in ASP(t)} ad(t,p,i) * x_{tp}\Big) \tag{4}$$

subject to:

$$\sum_{p \in ASP(t)} x_{tp} = 1 \ \ \forall t \in E_R \tag{5}$$

$$x_{tp} \in \{0, 1\} \tag{6}$$

Objective (4) is to maximize the minimum amount of demand serviced on a specific period of the time horizon. Constraint (5) requires that each task is assigned to exactly one pattern. Each decision variable x_{tp} equals to 1 if task t is serviced with pattern p, and 0 otherwise.

To solve the DBP, we devise an effective algorithm called DBTS based on the well-known tabu search method [13]. DBTS starts from a random solution which is generated as follows. For each task t, a service pattern is randomly selected from the allowable pattern set and assigned to the task. Then DBTS iteratively moves from the incumbent solution to one of its neighbor solutions. DBTS relies on a FLIP operator which for a given task, simply selects another pattern in its allowable service pattern set to replace the current pattern. At each iteration, DBTS applies the best move among the set of eligible moves (a move being eligible if it is non-tabu or if it leads to a neighbor solution better than all visited solutions). The involved task of the performed move, say u, is added to the so-called tabu list and cannot be changed for the next tt iterations (tt is a parameter called 'tabu tenure'). DBTS terminates either when the iteration counter reaches a predefined maximum number $MaxIter$ or the objective value reaches its best possible value $Best\_Obj$, calculated as $Best\_Obj = TD/m$ where $TD = \sum_{t \in E_R} \sum_{i \in H} d_i(t)$ is the total demand of all tasks over the time horizon.

DBTS outputs an assignment of service patterns to tasks such that the tasks are approximately evenly allocated over the time horizon H. Then, for each period $i \in H$, we run the path scanning algorithm [14] to solve the associated CARP to generate a number of routes able to cover all tasks assigned to period i. Basically, this algorithm always selects an arc (from a candidate set) that is the closest to the end of the route in construction. When an arc is selected, its inverse arc is removed from the candidate set to make sure that the required edge is served no more than once. More details about the implementation of the path scanning algorithm can be found in the original paper [14]. The whole set of resulting routes over all m periods constitutes the starting PCARP solution S_0 of the second phase of the HLS algorithm. The number of vehicles of this starting solution ($F_v(S_0)$) provides an upper bound which is used to prune any solution with a number of vehicles greater than $F_v(S_0)$.

3.2 Second phase: iterated improvement with pattern and route adjustments

The second phase of HLS seeks to further reduce the number of used vehicles and the total route cost. For this purpose, HLS jointly applies a pattern improvement procedure (PI) and a route improvement procedure (RI) combined with a perturbation mechanism to escape local optimum traps.

3.2.1 Pattern improvement

The PI procedure improves the solution by adjusting the pattern assignment of each task, which is achieved by exploring a neighborhood induced by the alter-pattern operator.

- **alter-pattern** (AP(t, p, q)): change the current service pattern p of task t to a new pattern q from the allowable service pattern set (i.e., $q \neq p$, $p, q \in ASP(t)$). To do this, AP removes, from the current solution, all the services of pattern p of task t, and then for each period of pattern q, inserts a new service to task t in the best location where the augmentation of the total costs is the least while the solution feasibility is maintained.

AP is a large-step-sized operator because it generates neighbor solutions that require multiple changes of the current solution. The number of changes depends on the number of periods in the dropped and new patterns. This operator allows a transition between structurally different feasible solutions which cannot be reached by the small-step-sized operators used in the route improvement procedure (see Sect. 3.2.2). From this point of view, PI and RI are two complementary procedures.

The PI procedure iterates on tasks in random order. For each task t, the patterns in the allowable service pattern set is examined one by one using the AP operator. Once an improving solution is found in terms of the problem objectives, the transition is performed before examining the next task. Given the current solution S and a new solution $S^{'}$, $S^{'}$ improves on S if $F_v(S^{'}) < F_v(S)$, or when $F_v(S^{'}) = F_v(S)$, $F_c(S^{'}) < F_c(S)$. The pattern improvement procedure stops when all tasks have been successively considered without leading to any solution improvement.

3.2.2 Route improvement

A PCARP solution S is composed of m CARP solutions where each CARP solution is a set of vehicle routes. During the route improvement (RI) procedure, the service pattern assignments to tasks are kept fixed and tasks are only allowed to displace within each period of the assigned service pattern. This amounts to solving m CARPs with the m CARP solutions of S as their starting solutions. To optimize a CARP solution related to a specific period, any approach to the conventional CARP can be applied in general. However, expensive CARP approaches like [4,21,30,31] are not suitable in our case due to two reasons. First, they might significantly increase the overall solution time since the CARP routine is frequently called in our HLS algorithm. Indeed, m calls of CARP approach are needed at each RI-PI iteration and this might continue for a large number of iterations. Second, a CARP approach able to provide optimal (or very high quality) CARP solutions does not necessarily lead to better PCARP results. This is because a PCARP solution is determined not only by the routing plan at each period but also by its pattern assignment. Allowing HLS to explore more pattern assignments instead of spending

a large portion of time in optimizing the routing plan to a given pattern assignment would presumably lead to better PCARP solutions.

Given the above remarks, we decided to call for a dedicated fast CARP heuristic able to provide good quality solutions. For this purpose, we adopted the Variable Neighborhood Search framework [16] to devise a simple and fast VNS procedure with constrained neighborhoods (VNS-C). We notice that the idea of VNS has previously been used in several effective CARP heuristics (e.g., [18, 29]). Our VNS-C procedure shares with these existing procedures a set of common move operators and makes improvements in particular by introducing a dedicated neighborhood reduction method. Our experimental experience showed that VNS-C is a suitable procedure for route improvement in our context and contributes to the overall high performance of the HLS algorithm. VNS-C relies on four traditional operators (neighborhoods) for the CARP, namely Single insertion (N_1), Double insertion (N_2), Swap (N_3) and Two-opt (N_4), which are described as follows [21].

Let u and v be a pair of tasks in the current CARP solution, tasks x and y be respectively the successor of u and v, $rt(u)$ be the route including task u.

- Single insertion (N_1): displace task u after task v (also before task v if v is the first task of $rt(v)$); both arcs of u are considered when inserting u in the target position, and the one yielding the best solution is selected;
- Double insertion (N_2): displace a sequence (u, x) after task v (also before task v if v is the first task of $rt(v)$); similar to N_1, both directions are considered for each task and the resulting best move is chosen;
- Swap (N_3): exchange task u and task v; similar to N_1, both directions are considered for each task to be swapped and the resulting best move is chosen;
- Two-opt (N_4): two cases exist for this move operator: 1) if $rt(u) = rt(v)$, reverse the direction of the sequence (x, v); 2) if $rt(u) \neq rt(v)$, cut the link between (u, x) and (v, y) , and establish a link between (u, y) and (v, x);

Notice that these operators test both arcs for each associated required edge, but only one of them is selected finally. This ensures that for each required edge, only one direction is served at any time of the route improvement process.

In order to accelerate neighborhood examinations, we use an estimation criterion to reduce the number of neighbor solutions to be considered, leading to a constrained version of the four neighborhoods. This estimation criterion is based on a distance measure between two tasks t_1, t_2 which is defined as:

$$D_{task}(t_1, t_2) = (\sum_{a=1}^{2} \sum_{b=1}^{2} D(v_a(t_1), v_b(t_2)))/4 \tag{7}$$

where $D(v_a(t_1), v_b(t_2))$ is the traversing distance between t_1's ath end node $v_a(t_1)$ and t_2's bth end node $v_b(t_2)$ (this distance measure was first used in [26] to define the distance between two routes). For each task t, we create a list $L_t(nl)$ that

contains t's nearest nl (nl is a parameter, $nl < n$) neighboring tasks according to formula (7). Then at each iteration, VNS-C restricts its examination to those moves involving at least one other task taken from the list $L_t(nl)$. One notices that all four move operators involve two distinct tasks. Indeed, *insertion* is to insert one task after or before another task; *swap* is to swap one task with another task; *two-opt* is to exchange a subsequent part of a task with that of another one. As such, given a task t, the other task the move operator is allowed to consider is delimited inside $L_t(nl)$, which leads to the neighbor solutions that VNS-C explores.

As outlined in Algorithm 2, at each iteration, VNS-C applies k times the $Shake$ operation, each $Shake$ application performing a move randomly chosen from N_3. Then a basic variable neighborhood descent (VND) based on the neighborhoods N_1–N_4 is applied to attain a local optimum $S^{'}$. If $S^{'}$ is better than the current solution S, S is replaced by $S^{'}$ and k is reset to its initial value 1; otherwise k is incremented by 1. This process continues until k reaches k_{max}.

Algorithm 2: VNS-C procedure

Data: S: a $CARP$ solution; k_{max}: max allowed iterations without improvement
Result: the best solution S^* found

```
S ← VND(S) ;
k ← 1 ;
while k < k_max do
    S' ← S ;
    for i = 1 to k do
        S' ← Shake(S', N_3) ;
    S' ← VND(S') ;
    if S' improves on S then
        S ← S' ;
        k ← 1 ;
    else
        k ← k + 1 ;
return S*
```

3.2.3 Perturbation to escape local optimum traps

When the current solution cannot be further improved by the PI and RI procedures, a local optimum is reached. HLS then triggers the *threshold based exploration* (TBE) procedure (see Algorithm 3) to escape the trap. TBE basically applies the alter-pattern operator of Section 3.2.1 with a solution accepting criterion. The choice of the alter-pattern operator is based on its two remarkable characteristics. First, AP is an inter-period operator able to modify multiple periods. Second, AP can simultaneously modify pattern assignments and routes in a single operation.

Different from the improvement procedures where the alter-pattern operator is used to seek strictly improving solutions, TBE accepts under specific conditions deteri-

orating solutions. In particular, TBE adapts the Threshold Accepting heuristic [10] to control the solution deterioration. TBE starts by a random shuffling of all tasks. For each task, TBE examines the patterns in the allowable service pattern set with the alter-pattern operator. A neighbor solution $S^{'}$ is accepted to replace the incumbent solution S if $F_v(S^{'}) < F_v(S)$, or when $F_v(S^{'}) = F_v(S)$, $F_c(S^{'}) <= \delta$. δ is a total-cost threshold value which is determined according to the current best total cost $F_c(S^*)$ and a ratio r (r is a parameter): $\delta = (1 + r) \times F_c(S^*)$.

Algorithm 3: Threshold based exploration procedure

Data: S: a $CARP$ solution; L: exploration strength; r: threshold ratio; S^*: input best solution
Result: the best solution S^* found

```
1  δ ← (1 + r) × F_c(S*) ;
2  for i = 1 to L do
3      Randomly shuffle all tasks in T ;
4      for t ∈ T do
5          for p ∈ ASP(t) do
6              S' ← apply alter-pattern operator and pattern p to task t of S ;
7              if (F_v(S') < F_v(S)) ∨ ((F_v(S') = F_v(S)) ∧ (F_c(S') <= δ)) then
8                  S ← S' ;
9              if (F_v(S') < F_v(S*)) ∨ ((F_v(S') = F_v(S)) ∧ (F_c(S') < F_c(S*))) then
10                 S* ← S' ;
11                 δ ← (1 + r) × F_c(S*) ;
12 return S and S*
```

4 Experimental Evaluation

To assess the proposed HLS algorithm, we show in this section computational results in comparison with the best performing algorithms in the literature based on three sets of 63 well-known PCARP benchmark instances.

4.1 Benchmarks

The PCARP benchmark instances used in our experiments are very popular and widely used in previous studies, which belong to three sets: *pgdb*, *pval* and *pG*. The *pgdb* and *pval* sets were generated from the corresponding *gdb* and *val* CARP benchmark sets. The *pG* set was extended from the G set generated by Brandão and Eglese [3], which consists of 10 large CARP instances based on a road network in Lancashire, U.K. The G set consists of two groups (G1 and G2), each with 5 instances (denoted as 1-A∼1-E and 2-A∼2-E). To ensure that the instances after extension have different capacities, the pG set excludes four instances and finally

contains 6 instances. These instances were generated to simulate a weekly waste collection. Therefore, the time horizon is one week, with Saturday and Sunday as idle days. If the service frequency of an edge (a street) is 2 or 3, then consecutive services over the horizon (e.g., (Monday, Tuesday) and (Monday, Tuesday, Wednesday)) are forbidden. For each required edge $e = (u, v)$ where u and v are numbered, the service frequency is defined as $f(e) = 1 + (u + v)\%5$, and each element $d_i(e)$ in the demand vector $d(e)$ is the demand of (u, v) in the original CARP instance. The capacities were duplicated (by 2, 3 or 4 depending on the instance) such that the accumulated demand never exceeds the vehicle capacity. These three sets cover instances of diverse scales from small-sized to large-sized, with a number of nodes ranging from 7 to 255, and a number of edges ranging from 11 to 375.

The optimal F_v values for most instances were previously known, except three instances whose optimal F_v values are proved for the first time in this work (see Section 4.4). This is achieved when our upper bound coincides with the lower bound (provided in [5]) for a given instance. On the other hand, optimal F_c values for these instances are still unknown.

4.2 Experiment setup

HLS was coded in C++ and compiled by GNU g++ 4.1.2 with '-O3' flag. The experiments were conducted on a computer with an AMD Opteron 4184 processor (2.8GHz and 2GB RAM) running Ubuntu 12.04. Running the DIMACS machine benchmark program [1] without compilation optimization flag on our machine to solve graphs r300.5, r400.5 and r500.5 requires 0.40, 2.50 and 9.55 seconds respectively.

For our comparative studies on *pgdb* and *pval* sets, the MARM algorithm [28] was used as our main reference since for these two sets, it dominates previous PCARP algorithms like LMA [22] and SS [6], producing all the best known results in terms of both F_v and F_c except one case (pgdb14). For the *pG* set, we used three reference algorithms: MARM, LMA and SS. These algorithms have different performances in terms of F_v and F_c for the *pG* instances. When comparing the average results, we only compare with MARM since it outperforms other existing approaches.

Following the practice of previous PCARP approaches, we ran our HLS algorithm 30 times to solve each instance and reported the best and average results [2]. The results of HLS are indicated in bold if they attain the current best known results. If HLS further improves on the current best known results (i.e., the results of MARM), the results are starred. The *#Bests* row counts the number of instances on which each algorithm has achieved the best result among all the compared results. Here-

[1] dfmax: ftp://dimacs.rutgers.edu/pub/dsj/clique/

[2] Our solution certificates will be available online

after, we confirm a solution value (either F_v or F_c) is optimal when our upper bound (the solution value provided by HLS) matches the lower bound (provided in [5]).

4.3 Parameter calibration

HLS relies on a number of parameters whose settings are summarized in Table 1. These settings were identified in the following manner. The parameters for the tabu search procedure of the first phase were tuned independently whose goal was to find an upper bound of F_v as small as possible. According to our experimental experience, we set the tabu tenure tt to a value linearly correlated to the number of tasks n plus rand(-2,2) (which denotes a random number between -2 and 2). *MaxIter* is set large enough to ensure the best outcome of the largest pG set. This large value of *MaxIter* does not lead to time waste for smaller instances since tabu search may terminate before attaining *MaxIter* iterations when F_v reaches its best possible value (see Section 3.1).

The rest of the parameters were tuned by the Iterated F-race (IFR) method [2, 24], which allows an automatic parameter configuration. IFR takes, for each parameter, a range of values as input. These ranges, listed in Table 1, were determined by preliminary experiments. We set the tuning budget to 2000 runs of HLS and formed the training set with 16 representative instances (4 from *pgdb*, 10 from *pval* and 2 from *pG*). To ease the use of IFR, we added an additional line of code in our HLS program to output a single solution value to IFR which aggregates the two objective values of the final solution by giving a large enough weight coefficient α to F_v (i.e., $f(S) = \alpha \times F_v(S) + F_c(S)$). It is obvious that this aggregation does not alter the tuning results. The final choice of the parameter values shown in Table 1 were used in all experiments in the following sections.

The parameters of tabu search were tuned separately since the first phase of HLS is independent of the second phase. Moreover, according to our experimental observations, tuning all parameters (including those of tabu search) via IFR requires more tuning budget while does not lead to better parameter settings.

Table 1
Parameter settings of HLS.

Component	Name	Description	Range	Value
Solution initialization	*tt*	tabu tenure	-	0.4n + rand(-2,2)
	MaxIter	max iteration	-	100000
Improvement stage	k_{max}	max number of iterations without improvement	[10,20]	15
	nl	neighbor list size	[6,16]	10
Perturbation stage	L	exploration strength	[20, 50]	30
	δ	threshold ratio	[0.01,0.1]	0.04
Overall	W	max allowed number of non-improving attractors	[5,20]	10

4.4 Experimental results

The comparative results of our HLS algorithm and the reference algorithms on the *pgdb*, *pval* and *pG* sets are summarized in Tables 2–4. From these three tables, we observe that HLS performs remarkably well on the three benchmark sets both in terms of the best and average performances over 30 runs.

When we consider the best performance (columns "Best results", Tables 2–4), we can make the following observations. First, HLS outperforms the state-of-the-art algorithm MARM on a large portion of the benchmark instances. Out of the total of 63 instances, HLS discovers 44 (69.8%) new best known results (new upper bounds), and matches the current best known result for other 8 cases. Second, HLS reaches the optimal F_v values for *all* test instances, where three of them (G1-B, G2-B and G2-C) are proved to be optimal for the first time by HLS (the optimality is confirmed since the upper bound value of F_v listed in Table 4 coincides with the lower bound value provided in [5]). Such a remarkable performance was not observed for previous PCARP algorithms. In particular, for the three large-sized instances of *pG* set where HLS manages to reach the optimal F_v value for the first time, HLS also obtains a smaller F_c value compared to MARM which demonstrates the dominance of HLS on these three instances. Third, for the remaining 60 instances where MARM also achieves the optimal F_v values, HLS finds 41 smaller and 8 equal F_c values compared to the best known solutions. Finally, we observe a clear trend that the superiority of our HLS algorithm over the reference algorithms becomes prominent as the size of instance increases. HLS improves and matches the best known results by 47.8% (11/23) and 69.6% (16/23) respectively for the small-sized instances in the *pgdb* set, by 79.4% (27/34) and 88.2% (30/34) for the medium-sized instances in the *pval* set while for the large-sized instances of the *pG* set, HLS improves all the best known results.

Now if we examine the average performance of our HLS algorithm (columns "Average results", Tables 2–4), we can make the following comments. First, HLS competes favorably with MARM on the small and medium sized instances. For the *pgdb* set, HLS matches MARM for 18 instances, and performs slightly worse for 5 cases. Among the instances with equal average F_v values, HLS improves 7 average F_c values compared to MARM. For the *pval* set, HLS performs marginally worse than MARM in terms of F_v for 4 instances. On the other hand, HLS outperforms MARM in terms of F_c for 25 out of the 30 instances where both algorithms achieve the same F_v values. Second, for the *pG* real-world data set, HLS consistently dominates MARM in term of both F_v and F_c for all 6 instances. For 3 out of 6 instances, even the average F_c results of HLS are better than the best results of MARM.

Finally, Table 5 compares the computational efficiency of HLS and MARM in terms of the average runtime (in seconds) on the three test sets. Since MARM and HLS were tested on different computers, we made a time conversion by scal-

Table 2
Comparative results (best and average) of our HLS algorithm with MARM on the 23 instances of the *pgdb* set. The results of HLS are in boldface if they are at least as good as the best known results. The results of HLS are starred if they improves on the best known results from MARM.

INST	$(\|V\|,\|E\|,\|S\|)$	Best results				Average results			
		MARM		HLS		MARM		HLS	
		F_v	F_c	F_v	F_c	F_v	F_c	F_v	F_c
1	(12,44,65)	3	810	**3**	815	3.00	827.40	**3.00**	841.33
2	(12,52,76)	3	917	**3**	**913***	3.00	934.60	**3.00**	939.07
3	(12,44,61)	3	691	**3**	**691**	3.00	710.40	**3.00**	717.80
4	(11,38,52)	2	740	**2**	747	2.00	762.10	**2.00**	772.93
5	(13,52,75)	3	1004	**3**	1022	3.00	1032.40	**3.00**	1046.60
6	(12,44,67)	3	900	**3**	**895***	3.00	912.20	**3.00**	917.50
7	(12,44,65)	3	819	**3**	**815***	3.00	836.00	**3.00**	848.97
8	(27,92,143)	5	953	**5**	**937***	5.00	986.00	5.10	**964.17***
9	(27,102,155)	5	892	**5**	**883***	5.00	917.30	**5.00**	**897.10***
10	(12,50,65)	2	677	**2**	681	2.00	696.30	**2.00**	701.00
11	(22,90,133)	3	1089	**3**	**1081***	3.00	1113.40	**3.00**	**1100.57***
12	(13,46,67)	3	1118	**3**	1142	3.00	1149.80	**3.00**	1173.80
13	(10,56,81)	3	1555	**3**	1557	3.00	1564.90	**3.00**	1568.80
14	(7,42,64)	2	290	**2**	**290**	2.50	291.00	2.70	**289.83***
15	(7,42,64)	2	174	**2**	**174**	2.00	176.20	**2.00**	**176.00***
16	(8,56,85)	3	360	**3**	**358***	3.00	364.20	**3.00**	**361.87***
17	(8,56,85)	2	261	**2**	**259***	2.00	266.10	2.07	**263.00***
18	(9,72,106)	2	487	**2**	**487**	2.00	494.40	**2.00**	**492.40***
19	(8,22,30)	2	171	**2**	**171**	2.00	172.70	**2.00**	173.47
20	(11,44,63)	2	348	**2**	350	2.00	357.80	2.67	**351.57***
21	(11,66,101)	3	498	**3**	**495***	3.00	504.90	**3.00**	**500.33***
22	(11,88,129)	4	589	**4**	**585***	4.00	593.00	**4.00**	**590.40***
23	(11,110,165)	5	686	**5**	**683***	5.00	691.40	5.10	**686.83***
#Bests		23	12	23	16	23	11	18	12

ing the CPU times of MARM reported in its original papers into its equivalent AMD Opteron 4184 2.8 GHz run times. Following [28], the conversion is based on the assumption that the processor speed is approximately linearly proportional to the processor frequency. Hence, the runtime of MARM is reduced by a factor of $2/2.8 = 0.71$. This comparison is only made for indicative purposes, since in addition of the processor, the runtime of MARM is also influenced by some inaccessible factors such as the operating system, compiler and coding skills of the programmer. Nevertheless, Table 5 provides interesting indications. Indeed, after time conversion, HLS is still always much faster than MARM in solving *pgdb* and *pval* data sets (in less than 10 seconds). For the real-world *pG* data set, HLS attains its solutions in about seven minutes (the runtime of MARM is unavailable).

5 Analysis and Discussions

In this section, we perform additional analysis to study the contribution of each underlying component of the proposed algorithm. Specifically, we explore the role of the first phase in enhancing the primary objective F_v, and the contribution of the

Table 3
Comparative results of our HLS algorithm with MARM on the 34 instances of the *pval* set. The results of HLS are in boldface if they are at least as good as the best known results. The results of HLS are starred if they improves on the best known results from MARM.

INST	$(\|V\|, \|E\|, \|S\|)$	Best results				Average results			
		MARM		HLS		MARM		HLS	
		F_v	F_c	F_v	F_c	F_v	F_c	F_v	F_c
1a	(24,78,105)	2	470	**2**	**470**	2.00	488.80	**2.00**	**488.70**
1b	(24,78,105)	3	530	**3**	**520***	3.00	540.00	**3.00**	**533.87**
1c	(24,78,105)	4	653	**4**	**652***	4.00	678.50	4.20	**667.27**
2a	(24,68,94)	2	697	**2**	699	2.00	706.00	**2.00**	711.00
2b	(24,68,94)	2	775	**2**	786	2.00	788.80	**2.00**	794.03
2c	(24,68,94)	4	1149	**4**	1169	4.00	1183.30	**4.00**	1190.33
3a	(24,70,96)	2	222	**2**	**218***	2.00	225.90	**2.00**	**223.80**
3b	(24,70,96)	2	255	**2**	256	2.00	263.80	**2.00**	264.70
3c	(24,70,96)	4	336	**4**	**336**	4.00	347.10	**4.00**	**346.77**
4a	(41,138,205)	2	1228	**2**	**1216***	2.00	1262.40	**2.00**	**1250.37**
4b	(41,138,205)	3	1288	**3**	**1254***	3.00	1314.70	**3.00**	**1288.60**
4c	(41,138,205)	4	1409	**4**	**1374***	4.00	1438.70	**4.00**	**1408.57**
4d	(41,138,205)	6	1858	**6**	**1853***	6.10	1905.60	6.47	**1864.83**
5a	(34,130,194)	2	1315	**2**	**1309***	2.00	1353.30	**2.00**	**1335.03**
5b	(34,130,194)	3	1384	**3**	**1381***	3.00	1417.60	**3.00**	**1395.67**
5c	(34,130,194)	4	1522	**4**	**1495***	4.00	1514.90	**4.00**	**1514.73**
5d	(34,130,194)	6	1991	**6**	**1960***	6.00	2033.30	6.17	**1988.10**
6a	(31,100,150)	2	722	**2**	**722**	2.00	741.40	**2.00**	**733.37**
6b	(31,100,150)	3	774	**3**	**758***	3.00	786.40	**3.00**	**772.07**
6c	(31,100,150)	7	1117	**7**	**1102***	7.00	1139.00	7.07	**1122.90**
7a	(40,132,201)	2	966	**2**	**960***	2.00	997.70	**2.00**	**984.20**
7b	(40,132,201)	3	960	**3**	**938***	3.00	978.50	**3.00**	**951.50**
7c	(40,132,201)	7	1165	**7**	**1135***	7.00	1190.80	**7.00**	**1149.93**
8a	(30,126,194)	2	1292	**2**	**1289***	2.70	1282.30	**2.13**	1313.57
8b	(30,126,194)	3	1301	**3**	**1271***	3.00	1330.40	**3.00**	**1305.13**
8c	(30,126,194)	7	1853	**7**	**1829***	7.00	1886.00	**7.00**	**1850.13**
9a	(50,184,274)	2	966	**2**	**937***	2.00	990.00	**2.00**	**957.47**
9b	(50,184,274)	3	990	**3**	**951***	3.00	1014.00	**3.00**	**968.20**
9c	(50,184,274)	4	1031	**4**	**995***	4.00	1050.90	**4.00**	**1003.73**
9d	(50,184,274)	7	1324	**7**	**1279***	7.00	1367.20	**7.00**	**1297.20**
10a	(50,194,300)	2	1385	**2**	**1356***	2.30	1401.80	**2.00**	**1380.63**
10b	(50,194,300)	3	1395	**3**	**1356***	3.00	1415.90	**3.00**	**1375.83**
10c	(50,194,300)	4	1461	**4**	**1417***	4.00	1477.10	**4.00**	**1437.80**
10d	(50,194,300)	7	1837	**7**	**1791***	7.00	1879.80	**7.00**	**1812.43**
#Bests		34	7	34	30	32	5	30	29

Table 4
Comparative results of our HLS algorithm with three state-of-the-art algorithms on the 6 instances of the *pG* set. The best of average results and best results are in boldface. The best result of HLS is starred if it is a new best known result.

INST	$(\|V\|, \|E\|, \|S\|)$	Best results								Average results			
		LMA		SS		MARM		HLS		MARM		HLS	
		F_v	F_c	F_v	F_c	F_v	F_c	F_v	F_c	F_v	F_c	F_v	F_c
G1-A	(255,347,1062)	11	3623482	12	3513307	10	3678504	**10**	**2984671***	11.00	3758286.10	**10.17**	**3020754.17**
G1-B	(255,347,1062)	14	3874996	15	3808003	13	3947359	**12***	**3297719***	14.00	4093805.00	**12.57**	**3335925.87**
G1-C	(255,347,1062)	13	3690919	14	3710681	11	3799614	**11**	**3149909***	12.30	3935364.70	**11.27**	**3175608.57**
G2-A	(255,375,1138)	12	3820365	14	3814253	11	3923292	**11**	**3236804***	12.30	4051017.50	**11.13**	**3266259.50**
G2-B	(255,375,1138)	15	4118258	16	4108672	14	4235707	**13***	**3533766***	15.00	4411075.30	**13.27**	**3563373.50**
G2-C	(255,375,1138)	14	4019023	15	3949113	13	4103535	**11***	**3524297***	13.40	4210976.60	**11.80**	**3431076.47**

Table 5
Average runtime (in CPU seconds) of HLS and MARM on the three test sets, na=not available.

Algo.	*pgdb*	*pval*	*pG*
MARM	8.7	37.3	na
HLS	2.1	9.0	431.1

three components of the second phase in optimizing the second objective F_c.

5.1 Role of the first phase

Given the priority of the first objective F_v, the very first phase of HLS aims to evenly allocate the tasks over the time horizon by solving a demand balancing problem. On the other hand, one notices that F_v can also be optimized by the second phase which is realized by an alternation between pattern improvements and route improvements. A natural question arises: is the first phase necessary? And if the answer is affirmative, which of the two phases provides more improvements of F_v? To answer these questions, we compare HLS with two HLS variants: RandCnst and 2nd_HLS. RandCnst constructs a random solution which starts with a random assignment of patterns to tasks and then applies the path scanning algorithm [14] to solve the CARP of each period. 2nd_HLS is a HLS variant without the first phase, which starts with a random solution provided by RandCnst, and then improves it with the second phase of HLS. This experiment allows us to appreciate how much the second phase is able to improve on a random solution in terms of F_v, and how much the first phase further contributes to the improvement of the first objective.

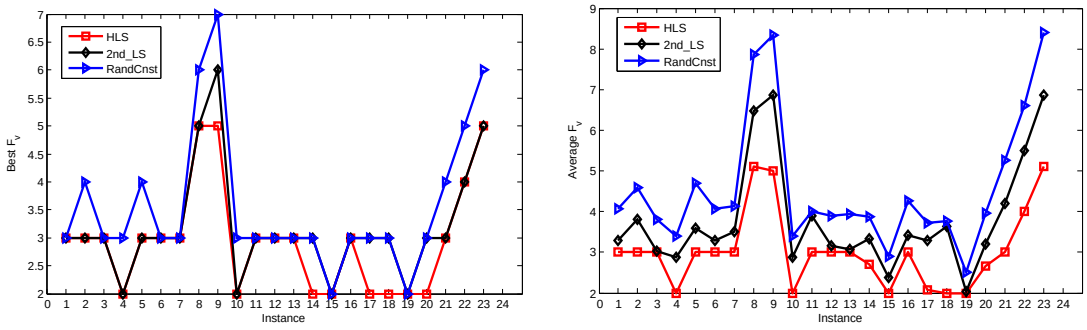
HLS and the two algorithm variants were run 30 times on the three sets of benchmarks. The best and average results (over 30 runs) in terms of F_v are plotted in Fig. 1-3 where instances are numbered according to the order they appear in Table 2-4. A Wilcoxon signed-rank test with a significance factor of 0.05 was also applied to detect the statistical difference between any two comparable samples, and the outcomes are displayed in Table 6 where for each comparison item, we listed the negative rank sum (R-), the positive rank sum (R+), the resulting p-value (p-value) and whether a significant difference is detected (diff?). From Fig. 1-3 and Table 6, we can make the following observations.

- First, HLS on average attains the best outcome, followed by 2nd_HLS and finally RandCnst. Indeed, the red lines (HLS) are generally below the black lines (2nd_HLS), which are further below the blue lines (RandCnst). This observation implies that 1) the second phase of HLS is able to optimize F_v in many cases; 2) when it is combined with the first phase, more significant improvements are achieved. The results of the Wilcoxon test (see Table 6) additionally confirm the above observations, which indicate in a statistical manner the superiority of HLS over 2nd_HLS, and of 2nd_HLS over RandCnst in a majority of cases. In partic-

ular, HLS easily dominates 2nd_HLS on all data sets in terms of both best and average results (since a p-value$<$0.05 is detected for all compared samples). We thus conclude that the first phase of HLS is undoubtedly necessary.

- Second, the improvement of HLS over 2nd_HLS generally enlarges as the instance size increases. Indeed, the extent of improvement of HLS over 2nd_HLS on the pG set (Fig. 3) is much larger than that on the $pgdb$ set (Fig. 1) and the $pval$ set (Fig. 2). This observation confirms the critical role of the first phase in ensuring the high performance of HLS, especially for large-sized problems. Also from Fig. 3, we can clearly remark that the first phase provides more improvement than the second phase in terms of F_v. Indeed, the gap between HLS and 2nd_HLS is much larger than the gap between 2nd_HLS and RandCnst.

The above experiments indicated that the performance of HLS deteriorates if its first phase is disabled. Meanwhile, it would be interesting to know how tight the upper bound of F_v obtained by the first phase is. For this purpose, we ran the first phase of HLS (denoted as 1st_HLS) 30 times on the three instance sets ($pgdb$, $pval$ and pG) and summarized the number of instances for which $\sigma = X$ ($X = 0, 1$) is satisfied in Table 7. For a given instance, $\sigma = X$ indicates that the gap between the worst F_v (obtained by 1st_HLS over 30 runs) and the best known F_v (obtained by HLS shown in Table 2-4) is X. From Table 7, we observe that the upper bound of F_v provided by 1st_HLS is very tight. Among all 63 test instances, the worst F_v is only one unit from the best known F_v. Moreover, for 39 instances (14 from $pgdb$ and 25 from pG) out of 63, the worst F_v is exactly the best known F_v, which means that 1st_HLS consistently produces the best known F_v over 30 runs.

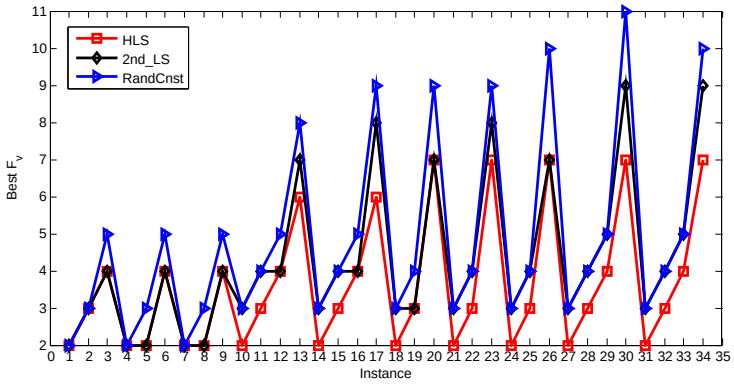


(a) Best Results (b) Average results

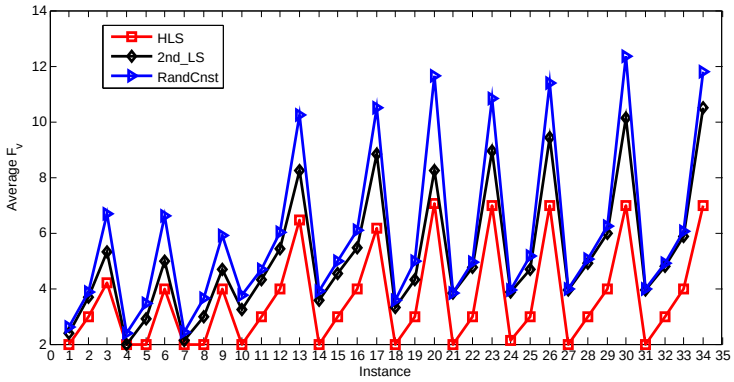
Fig. 1. F_v results of three algorithm variants on the *pgdb* set

5.2 Contribution of the second phase

The improvement of the second objective F_c mainly relies on the second phase of HLS, which involves three algorithm components – the route improvement (RI) procedure, the pattern improvement (PI) procedure and the perturbation procedure. In this section, we set the route improvement procedure as the base and investigate

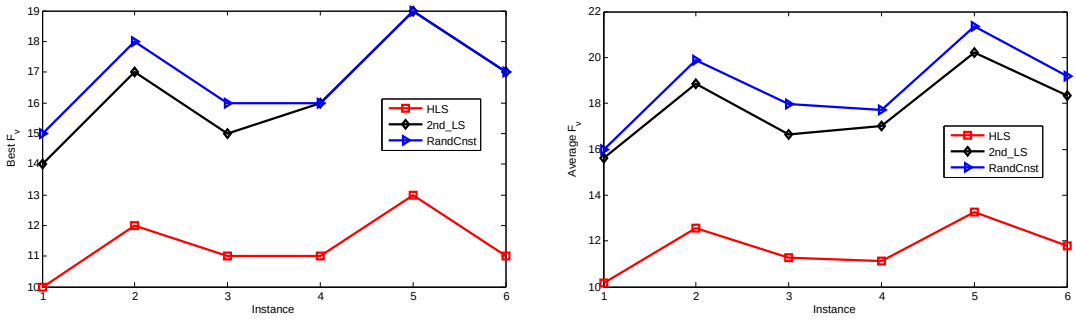


(a) Best Results



(b) Average results

Fig. 2. F_v results of three algorithm variants on the *pval* set



(a) Best Results (b) Average results

Fig. 3. F_v results of three algorithm variants on the *pG* set

the contribution of the other two components in optimizing F_c. For this purpose, we compare HLS to two algorithm variants: HLS_1 and HLS_2. HLS_1 is a HLS variant without PI and perturbation. HLS_2 includes PI, but without perturbation. Furthermore, we rename the original HLS algorithm as HLS_3. The three algorithms were run 10 times on 8 representative instances (see Table 8). For each instance, we report the results according to two indicators: Δ_{12} and Δ_{23}, which are calculated as follows. $\Delta_{12} = (avg_1 - avg_2) \times 100/avg_3$, where avg_1 and avg_2 denote the average F_c (over 10 runs) attained by HLS_1 and HLS_2 respectively. This indicator reports

Table 6
Wilcoxon test for pairwise comparison between the best and average results obtained by three algorithm variants (HLS, 2nd_HLS and RandCnst) in terms of F_v.

Algorithm Pair	DataSet	Best Results				Average Results			
		R-	R+	p-value	Diff?	R-	R+	p-value	Diff?
HLS v.s 2nd_HLS	*pgdb*	15	0	3.69e-02	Yes	276	0	2.88e-05	Yes
	pval	210	0	2.51e-05	Yes	561	0	5.64e-07	Yes
	pG	21	0	3.45e-02	Yes	21	0	3.13e-02	Yes
2nd_HLS v.s RandCnst	*pgdb*	45	0	3.35e-03	Yes	276	0	2.87e-05	Yes
	pval	120	0	3.26e-04	Yes	561	0	5.63e-07	Yes
	pG	6	0	0.15	No	21	0	0.03	Yes

Table 7
Results of the first phase of HLS (1st_HLS) that summarize the number of instances for which $\sigma = X$ ($X = 0, 1$) is satisfied according to the dataset. For a given instance, $\sigma = X$ indicates the gap between the worst F_v (obtained by 1st_HLS over 30 runs) and the best known F_v (obtained by HLS shown in Table 2-4).

Dataset	$\sigma = 0$	$\sigma = 1$
pgdb	14/23 (60.87%)	9/23 (39.13%)
pval	25/34 (73.53%)	9/34 (26.47%)
pG	0/6 (0.00%)	6/6 (100.00%)

the improvement of HLS_2 over HLS_1 in terms of average F_c which highlights the contribution of the PI procedure. Similarly, $\Delta_{23} = (avg_2 - avg_3) \times 100/avg_3$ reports the improvement of HLS_3 over HLS_2 and highlights the contribution of the perturbation procedure. The results are summarized in Table 8. The reason why we tested the algorithms 10 times on 8 representative instances is based on the following consideration. In order to investigate the effect of algorithm components on F_c, we should keep F_v unchanged in the second phase. For this purpose, we run each algorithm multiple times (more than 10) to select 10 runs where the algorithm attains the lower bound of F_v in the first phase, such that its unique mission in the second phase is to optimize F_c. This requirement is hard to meet in the general case and the 8 representative instances we selected are relatively easy for the algorithms to find such 10 runs.

From Table 8, we can see that the pattern improvement procedure and the perturbation procedure are two necessary and effective components in optimizing F_c. Indeed, by including the PI procedure to HLS_1, the improvement Δ_{12} reaches a significant average value of 7.711% in terms of F_c. Integrating the perturbation procedure to HLS_2 further leads to an improvement 2.857% on average. Clearly, compared to the perturbation procedure, the contribution of the PI procedure is even higher (on average 7.711% v.s 2.857%). We also applied a Wilcoxon test (with a significance factor of 0.05) to the pairwise comparison of HLS_1 v.s HLS_2 and HLS_2 v.s HLS_3, and the resulting p-value of 0.007813 (the same for both pairs) confirms that the improvements provided by the PI procedure and the perturbation procedure are significant in a statistical sense.

Table 8
Contribution of each algorithm component in the second phase on 8 representative instances. Δ_{12} indicates the improvement of HLS_2 over HLS_1 and highlights the contribution of the pattern improvement procedure. Δ_{23} indicates the improvement of HLS_3 over HLS_2 and highlights the contribution of the perturbation procedure.

Set	Instance	$\Delta_{12}(\%)$	$\Delta_{23}(\%)$
pgb	2	6.023	3.879
	9	4.494	2.922
	16	2.595	1.712
pval	1b	11.255	3.827
	4c	8.545	2.543
	6b	8.814	2.104
	9d	7.147	3.073
pG	G2-A	12.815	2.795
Average		7.711	2.857

6 Conclusions

The periodic capacitated arc routing problem is a relevant model with interesting applications. The PCARP has a primary objective of minimizing the number of vehicles and a secondary objective of minimizing the total cost. Motivated by the fact that existing PCARP methods rely on an aggregated weight function to optimize the two objectives, which has the clear drawback of examining many irrelevant candidate solutions, we devised the first approach using a two phased paradigm to focus on the primary objective to determine an upper bound for the number of vehicles during the first phase, and then use this bound to strongly constrain the search during the second phase by eliminating many irrelevant candidate solutions. To ensure the search efficiency, both search phases integrate problem specific search strategies and operators able to make a suitable balance between intensification and diversification of the search process.

Experimental assessments on three sets of 63 PCARP benchmark instances commonly used in the literature demonstrated the effectiveness of the proposed approach compared to the state of the art methods. In particular, HLS appears to be the first algorithm that is able to reach all the optimal values in terms of F_v, and establishes 44 improved new upper bounds in terms of F_c.

Furthermore, we showed that the first phase of the proposed HLS algorithm plays a critical role in optimizing the primary objective F_v, while the pattern improvement procedure and the perturbation procedure are two highly effective components of the second phase in improving the second objective F_c.

This study indicates that for a hierarchical optimization problem like the PCARP, it is more interesting to handle the hierarchical objectives as per and for each objective, it is relevant to design dedicated search procedures with respect to each

targeted optimization objective.

Acknowledgment

We are grateful to the reviewers whose constructive comments have helped us to improve the work. The work was partially supported by the PGMO project (2014-024H, Gaspard Monge Program for Optimization and Operations Research) and the National Natural Science Foundation of China (Grants 61473301, 71201171, 71501179, 71501180).

References

[1] P. Beullens, L. Muyldermans, D. Cattrysse, D. Van Oudheusden, A guided local search heuristic for the capacitated arc routing problem, European Journal of Operational Research 147 (3) (2003) 629–643.

[2] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated f-race: An overview, in: Experimental methods for the analysis of optimization algorithms, Springer, 2010, pp. 311–336.

[3] J. Brandão, R. Eglese, A deterministic tabu search algorithm for the capacitated arc routing problem, Computers & Operations Research 35 (4) (2008) 1112–1126.

[4] Y. Chen, J.-K. Hao, F. Glover, A hybrid metaheuristic approach for the capacitated arc routing problem, European Journal of Operational Research 253 (1) (2016) 25–39.

[5] F. Chu, N. Labadi, C. Prins, The periodic capacitated arc routing problem linear programming model, metaheuristic and lower bounds, Journal of Systems Science and Systems Engineering 13 (4) (2004) 423–435.

[6] F. Chu, N. Labadi, C. Prins, A scatter search for the periodic capacitated arc routing problem, European Journal of Operational Research 169 (2) (2006) 586–605.

[7] J.-F. Cordeau, M. Gendreau, G. Laporte, A tabu search heuristic for periodic and multi-depot vehicle routing problems, Networks 30 (2) (1997) 105–119.

[8] M. Díaz-Madroñero, D. Peidro, J. Mula, A review of tactical optimization models for integrated production and transport routing planning decisions, Computers & Industrial Engineering 88 (2015) 518–535.

[9] L. M. Drummond, L. S. Ochi, D. S. Vianna, An asynchronous parallel metaheuristic for the period vehicle routing problem, Future generation computer systems 17 (4) (2001) 379–386.

[10] G. Dueck, T. Scheuer, Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing, Journal of computational physics 90 (1) (1990) 161–175.

[11] P. Francis, K. Smilowitz, Modeling techniques for periodic vehicle routing problems, Transportation Research Part B: Methodological 40 (10) (2006) 872–884.

[12] M. Gaudioso, G. Paletta, A heuristic for the periodic vehicle routing problem, Transportation Science 26 (2) (1992) 86–92.

[13] F. Glover, M. Laguna, Tabu Search?, Springer, 2013.

[14] B. L. Golden, J. S. DeArmon, E. K. Baker, Computational experiments with algorithms for a class of routing problems, Computers & Operations Research 10 (1) (1983) 47–59.

[15] B. L. Golden, R. T. Wong, Capacitated arc routing problems, Networks 11 (3) (1981) 305–315.

[16] P. Hansen, N. Mladenović, J. Brimberg, J. A. M. Pérez, Variable neighborhood search, Springer, 2010.

[17] A. Hertz, G. Laporte, M. Mittaz, A tabu search heuristic for the capacitated arc routing problem, Operations research 48 (1) (2000) 129–135.

[18] Hertz A., Mittaz M, A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. Transportation science 35(4) (2001) 425–434.

[19] A. Kansou, A. Yassine, Ant colony system for the periodic capacitated arc routing problem, in: Proceedings of the 2009 International Network Optimization Conference, 2009, pp. 1–7.

[20] P. Lacomme, C. Prins, W. Ramdane-Chérif, Evolutionary algorithms for multiperiod arc routing problems, in: Proc. of the 9th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based systems (IPMU 2002), 2002, pp. 845–852.

[21] P. Lacomme, C. Prins, W. Ramdane-Chérif, Competitive memetic algorithms for arc routing problems, Annals of Operations Research 131 (1-4) (2004) 159–185.

[22] P. Lacomme, C. Prins, W. Ramdane-Chérif, Evolutionary algorithms for periodic arc routing problems, European Journal of Operational Research 165 (2) (2005) 535–553.

[23] H. Longo, M. P. de Aragão, E. Uchoa, Solving capacitated arc routing problems using a transformation to the cvrp, Computers & Operations Research 33 (6) (2006) 1823–1837.

[24] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, M. Birattari, The irace package, iterated race for automatic algorithm configuration, Tech. rep., Citeseer (2011).

[25] R. Martinelli, M. Poggi, A. Subramanian, Improved bounds for large scale capacitated arc routing problem, Computers & Operations Research 40 (8) (2013) 2145–2160.

[26] Y. Mei, X. Li, X. Yao, Cooperative coevolution with route distance grouping for large-scale capacitated arc routing problems, Evolutionary Computation, IEEE Transactions on 18 (3) (2014) 435–449.

[27] Y. Mei, K. Tang, X. Yao, A global repair operator for capacitated arc routing problem, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 39 (3) (2009) 723–734.

[28] Y. Mei, K. Tang, X. Yao, A memetic algorithm for periodic capacitated arc routing problem, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 41 (6) (2011) 1654–1667.

[29] M. Polacek, K.F. Doerner, R.F. Hartl, V. Maniezzo, A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. Journal of Heuristics 14(5) (2008) 405–423.

[30] L. Santos, J. Coutinho-Rodrigues, J. R. Current, An improved ant colony optimization based algorithm for the capacitated arc routing problem, Transportation Research Part B: Methodological 44 (2) (2010) 246–266.

[31] K. Tang, Y. Mei, X. Yao, Memetic algorithm with extended neighborhood search for capacitated arc routing problems, Evolutionary Computation, IEEE Transactions on 13 (5) (2009) 1151–1166.