# Evaluating Hyperheuristics and Local Search Operators for Periodic Routing Problems

Yujie Chen[1(✉)], Philip Mourdjis[1], Fiona Polack[1], Peter Cowling[1], and Stephen Remde[2]

[1] YCCSA, Computer Science Department, University of York, York, UK
{yc1005,pjm515,fiona.polack,peter.cowling}@york.ac.uk
[2] Gaist Solutions Limited, Lancaster, UK
stephen.remde@gaist.co.uk

**Abstract.** Meta-heuristics and hybrid heuristic approaches have been successfully applied to Periodic Vehicle Routing Problems (PVRPs). However, to be competitive, these methods require careful design of specific search strategies for each problem. By contrast, hyperheuristics use the performance of low level heuristics to automatically select and tailor search strategies. Hyperheuristics have been successfully applied to problem domains such as timetabling and production scheduling. In this study, we present a comprehensive analysis of hyperheuristic approaches to solving PVRPs. The performance of hyperheuristics is compared to published performance of state-of-the-art meta-heuristics.

**Keywords:** Hyperheuristic · Computational analysis · PVRP

## 1 Introduction

Most (meta-)heuristic approaches applied to new search problem domains need expert input in design. To automatize the process, hyperheuristics provide a problem-independent approach that automatically applies an appropriate search strategy, by calling low level heuristics (LLHs) at each decision point [1]. Whilst various hyperheuristics have been tested for a range of optimization problems (e.g. [1–4]), none has yet addressed PVRPs.

Hyperheuristic approaches operate at a management level, consisting of selection and acceptance stages. The LLH(s) to test at each search stage may be deterministically or probabilistically selected. In more advanced selection strategies, hyperheuristics can learn from the performance of previous selections; performance is usually evaluated using problem-independent measures such as change in the solution quality or elapsed CPU time. Acceptance determines whether to replace the current solution with the one yielded by the selected LLH(s). A large variety of acceptance strategies, such as only improved (OI) (e.g. [1]) and simulated annealing (e.g. [5]), have been tested in literature. Theoretically, a hyperheuristic should adapt to any hard computational search problem, and provides a mechanism for studying the strengths and weaknesses of LLHs for a specific problem.


F. Chicano et al. (Eds.): EvoCOP 2016, LNCS 9595, pp. 104–120, 2016.
DOI: 10.1007/978-3-319-30698-8_8

In this paper, we provide a comprehensive analysis on three types of hyperheuristics and apply them to benchmark and real-world PVRP instances with different characteristics. We also use the hyperheuristics to explore the strengths and weaknesses of LLHs designed for PVRPs. Section 2, briefly introduces PVRP and (meta-)heuristic solvers from literature. We then review LLHs designed for PVRP in Sect. 3. Section 4 presents the hyperheuristics. Sections 5 and 6 present the experimental design and analysis of the experiments respectively.

## 2 Periodic Routing Problem

PVRPs [6–9] provide a well-researched mathematical model for real-world problems such as inventory servicing, periodic maintenance, and on-site service planning. A PVRP comprises $K$ vehicles which can be used to service the demands of $N$ customers over $M$ days. Each PVRP has constraints that must be met by legal solutions: vehicles start and end their journey at a depot; no more than $K$ routes are built each day; vehicle capacity restrictions are respected; each customer request is serviced in one time slot by one vehicle; only one service pattern is chosen for each customer. A feasible visit pattern, $\lambda_i \in \Lambda_i$, for a customer $i$, is a pattern that meets all constraints and provides the level of servicing required for customer $i$. For example, a customer might require two service visits per week, on either Monday and Thursday or Tuesday and Friday, giving two feasible patterns. The PVRP objective is to design a set of daily routes, comprising feasible patterns for each customer, that minimizes the total travelling cost and satisfies the PVRP constraints.

### 2.1 Existing (Meta-)heuristic Solvers for PVRP

Heuristic approaches to PVRP developed since the 1970s [6,10–13] generate solutions by determining customer-day patterns that group geographically close customers. In 1995, Chao et al. [7] introduced a record-to-record meta-heuristic that outperformed the earlier heuristics approaches. Subsequent meta-heuristics approaches, including tabu search [8], scatter search [14] and variable neighbourhood search (VNS) [5], have all produced new best solutions for benchmark problems. Hybrid heuristics now present very competitive results: Gulczynski et al. [9] use integer programming-based improvement heuristics combined with routing-based local searches; Vidal et al. [15] propose a hybrid genetic algorithm that combines local search and sophisticated population management strategies to guide the search – an approach shown to perform better than all the above algorithms. Cordeau and Maischberger [16] combine tabu search and iterated local search to give a competitive, broad exploration of the search space.

## 3 Low Level Heuristics for the PVRP

A hyperheuristic has a repository of LLHs that operate directly on the solution space, and should provide good coverage of the solution space.

### 3.1 Constructive Heuristics

To construct a valid PVRP initial solution, most PVRP solvers first assign customers to days, then build a solution of a vehicle routing problem (VRP) for each day. For assignment, Cordeau et al. [8,16], Hemmelmayr et al. [5] and Vidal et al. [15] randomly select a feasible customer-day pattern for each customer; Chao et al. [7] and Gulczynski et al. [9] minimize the maximum demand serviced each day; Christofides and Beasley [6] minimize the daily total distance from each customer to the depot. To construct daily routes, the Clarke and Wright algorithm (CW) [17] and GENI insertion heuristic [18] are generally applied. Our hyperheuristics use the same approach as [5]: random assignment followed by a CW routes construction process for each day.

### 3.2 Perturbation Operators

From an initial solution, a hyperheuristic manages the application of perturbation operators to either daily routes or customer patterns. Application may be *first improvement* (FI) – seeking to improve the current solution, or *mutation* (shaking) to derive a similar, but new solution from the current solution.

**Route Related Operators.** There are a number of common operators used to modify single and multiple routes in PVRP (see VRP local search library [19]).

1. 2Opt: replace two edges from a route with two new edges (e.g. [7,9,15])
2. 3Opt: replace three edges from a route with three new edges (e.g. [5]).
3. Or-opt: remove a string of two to four nodes and insert it into a new position, either in the same route (e.g. [14]) or in a different route (e.g. [15]).
4. One point move (1PM): relocate a point to a new position, either in the same route or in a different route (e.g. [7,9]).
5. Two points swap (2PS): swap two points, either in the same route or between different routes (e.g. [9]).
6. Relocate: relocate a string of points from one route to another (e.g. [5,15]).
7. Cross: swap two chains of points between two routes. ([5,14]).

Route-based perturbation is typically applied as FI, embedded in an iterative local search (ILS) [7,9,14,15]; however, they can also be used as mutation operators: Hemmelmayr [5] uses "Relocate" and "Cross", for this purpose.

**Pattern Related Operators.** All these operators assign different valid patterns for selected customers. A customer with a new pattern is removed from their current routes and re-inserted to their new lowest-cost position on each day in the new pattern, meaning that we always get a complete PVRP solution.

1. Random pattern reassign (Pa_RR): randomly assign a new feasible visit pattern to $n$ customers drawn at random. A tabu mechanism prevents a customer from being subject to reassignment again in the short term.

2. Score based reassign (Pa_SR): chooses $n$ random customers, for each $i \in n$, assign the pattern $\lambda_i$ with the highest score, $Q(\lambda_i)$. $Q(\lambda_i)$ is updated after any pattern related operator use; if no improvement is found, $Q(\lambda_i) = \sqrt{Q(\lambda_i)}$.
3. Pattern reassign FI (Pa_FIR): for each customer $i$, successively test each feasible pattern, the first improvement found is executed.
4. Two points pattern swap (Pa_2SW): swaps the visit patterns of two customers $i$ and $j$ which have the same available patterns, $\Lambda_i = \Lambda_j$ and $\lambda_i \neq \lambda_j$.

**Mixed Operators.** To improve flexibility, mixed operators support moves between days, and potentially modify both routes structure and customer patterns. Chao et al. [7] propose an operator that removes the current routing of a customer's current pattern, and inserts into a different set of routes, with or without changing the customer's current pattern. For our LLH repository, we design two mixed operators that operate on chains; all customers moved must have the same available patterns, and only customers with a single visit per pattern are used.

1. Relocate with Pattern (MRPa): relocate a chain of points from one route to another route in the same day or a different day.
2. Cross with Pattern (MCPa): swap two chains of points between two routes within the same day or between days.

### 3.3 Reinitialization

If the current solution has not been improved for a certain number of iterations, we assume the search is stuck in a local optimum that cannot be escaped by a small mutation. A reinitialization mechanism, Algorithm 1, is applied. The new solution is made feasible by repeatedly removing the customer with the greatest load requirement from any route in the candidate solution that violates duration or load constraints, and re-inserting in a route where the constraints are met.

**Algorithm 1.** Reinitialisation

**Define**:
  $x_{best}$ is the best found solution so far
  $p_{random}$ is the probability of generating a random initial solution
**Reinitialisation**($x_{best}$, $p_{random}$)
**if** random(seed) $< p_{random}$ **then**
  random assignment and CW daily routes construction (Sect. 3.1).
**else**
  Destroy $w$% of the longest routes in $x_{best}$.
  For each customer in destroyed routes, randomly reassign feasible visit pattern.
  Insert each customer greedily to cheapest position in each day of assigned pattern.
**end if**
Return the new (re)constructed solution $x$.

## 4 Hyperheuristics

We consider three types of hyperheuristics: simple hyperheuristics, learning based hyperheuristics and VNS based methods.

### 4.1 Simple Hyperheuristics

Simple hyperheuristics [1] have basic LLH selection mechanisms such as simple random (SR), random descent (RD), random permutation, random permutation descent, and greedy. Acceptance strategies, such as all moves or only improving (OI) were originally tested on a sales summit problem [1]. Here, we implement SR and RD combined with OI acceptance, designated SROI and RDOI, respectively.

SR randomly selects a LLH, based on a uniform distribution. RD randomly selects a LLH and applies it repeatedly until there is no further improvement in the solution. OI accepts a new solution only if it is better than the current solution, evaluated by fitness.

### 4.2 Learning Based Hyperheuristics

Learning based hyperheuristics adapt the LLH trial set according to the historical performance of each LLH. In each iteration, a favourable LLH is applied, based on predefined rules: in our implementation, the LLH from the trial set that produces the most improved solution is applied. Three well known learning based selection mechanisms are tested: binary exponential back off (BEBO) [3], reinforcement learning (RL) [20] and a ranked choice function (CF) [1].

**BEBO.** [3] uses a tabu based learning mechanism. The tabu tenure, $tabu_i$, changes dynamically, such that a LLH that performs poorly is disabled for a number of iterations (which increases exponentially if the LLH subsequently perform poorly). Each iteration only LLHs $i$ with $tabu_i = 0$ are selected to form the trial set, $T$.

**RL.** [20] uses positive reinforcement to reward good LLH choices and negative reinforcement to penalise bad LLH choices. The utility value of each LLH is dynamically updated based on its performance, and the $w\%$ of LLHs with the highest utility form the trial set, $T$, for the next iteration. We apply hyperheuristic RL methods identified by Nareyek [20]. For each $LLH_i \in T$, $utility_i = \sqrt{utility_i}$. After testing, the best performing $LLH_{best} \in T$ that improves the solution is rewarded by setting $utility_i = utility_i^2 + 1$.

**CF.** [1] provides a different utility adoption scheme. In each iteration, the utility of each $LLH_i \in T$ is updated based on a linear function that considers the LLH's performance (evaluated by fitness change and execution time), the ability of the LLH in collaboration (evaluated by successively applied pairs of LLHs), and the elapsed time since the LLH was last called.

### 4.3 General Variable Neighbourhood Search (GVNS) with Learning

GVNS [21,22] differs from the hyperheuristics above in its more intensive use of local search (LS): the selected LLH is applied repeatedly rather than for one iteration only. Shaking is another critical component for GVNS, meaning that when no further improvement is found using LS, mutation operators are applied to facilitate the search to jump out of a local optimum.

**Algorithm 2.** General VNS

**Define:** $k_{max} = |LLH_{MU}|$, the number of mutation operators
**GVNS**$(x, LLH_{MU}, LLH_{FI}, t_{max})$

**while** $t < t_{max}$ **do**
  $k = 1$
  **while** $k < k_{max}$ **do**
    $x' = shaking(x, LLH^k_{MU})$
    $x'' =$ **VND**$(x', LLH_{FI})$
    **If** $x''$ is better than $x$ **then**
    $x = x''$ and $k = 1$
    **Otherwise** $k = k + 1$
  **end while**
**end while**

**Algorithm 3.** General VNSr

**GVNSr**$(x, LLH_{MU}, LLH_{FI}, t_{max})$
**while** $t < t_{max}$ **do**
  randomly choose $LLH^k_{MU}$ from $LLH_{MU}$
  $x' = shaking(x, LLH^k_{MU})$
  $x'' =$ **VND**$(x', LLH_{FI})$
  **If** $x''$ is better than $x$ **then**
  $x = x''$
**end while**

GVNS is a parameter-free approach; LLH selection uses a pre-ordered LLH set [5]. Our experiments need to test a large number of LLHs, which is very CPU-intensive. We propose variations to the GVNS: Algorithms 2 and 3 with, respectively, fixed order and random selection strategies to manage the selection of mutation operators ($LLH_{MU}$). The first stage of GVNS takes a candidate solution, $x$, and shakes it by applying one mutation operator (from $LLH_{MU}$). The second stage of GVNS calls a variable neighbourhood descent algorithm, VND, which applies FI operators ($LLH_{FI}$) to the shaken solution. VNS is run over a fixed time, $t < t_{max}$.

Algorithm 4 describes the VND procedure called in Algorithms 2 and 3. The VND manages selection of FI operators using either random ordering or one of three RL-based LLH orderings: ascending, descending, and top $w$%. Although utility is calculated in all cases, it is not used in random ordering selection.

## 5 Experimental Design

We design experiments that allow us to analyse the performance of hyperheuristics from different angles. We use data (benchmark and real) with different

**Algorithm 4. VND**$(x, LLH_{FI})$

Choose subset $LLH'_{FI} \subseteq LLH_{FI}$ based on the LLH selection approach used.
**Define:** $k_{max} = |LLH'_{FI}|$, the number of local search operators to be tested.
$k = 1$
**while** $k < k_{max}$ **do**
  $x' = \mathbf{ILS}(x, LLH^k_{FI})$, where **ILS** applies the selected FI operator, $LLH^k_{FI}$, repeatedly until no further improvement occurs. In each iteration of **ILS**, $utility_k = \sqrt{utility_k}$; if $LLH^k_{FI}$ makes an improvement, then $utility_k = utility_k^2 + 1$.
  **If** $x'$ is better than $x$ **then** $x = x'$ and $k = 1$
  **Otherwise** $k = k + 1$
**end while**

spatial characteristics. We also compare the performance of hyperheuristics with that of meta-heuristics applied to the benchmark problems.

The experiments are designed to replicate benchmark conditions from [15]. In particular, the search is always terminated after the fixed amount of CPU time stated in [15]. To check the suitability of this time limit for scalability experiments (Sect. 6.3), we ran preliminary experiments using twice the CPU time. We found no significant change in the quality of solutions, suggesting that a performance plateau is attained, and the chosen CPU time is appropriate.

All experiments are implemented in C♯ and executed on a cluster composed of 8 Windows computers, each with Intel Xeon E3-1230 CPU.

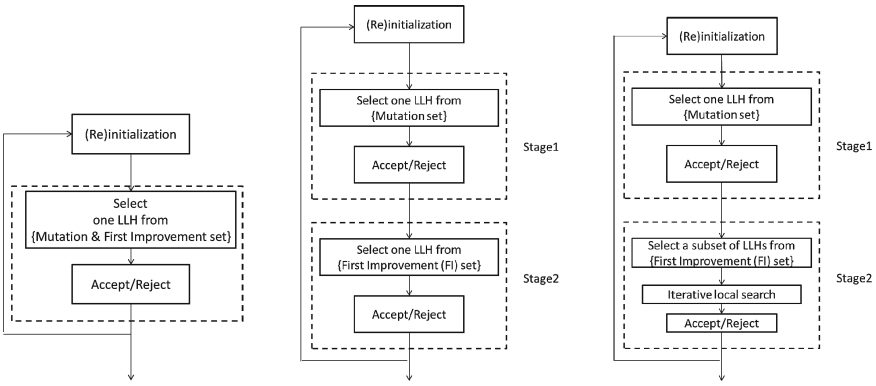
**LLH Repository Settings.** The operators introduced in Sect. 3 are classified according to whether we use them for mutation and/or FI, Table 1. Route related operators are parametrized by route ID, day, length of chain and number of points changed in one move; this makes it possible for an intelligent hyperheuristic to select a LLH specifically related to each sub-problem (e.g. daily

**Table 1.** LLH Repository used in our PVRP hyperheuristics

| Type | Operators |
|---|---|
| Route related: mutation | 2 points swap (2PS), Relocate, Cross |
| Route related: FI | 2Opt, 3Opt, 2PS, Relocate, Cross |
| Pattern related: mutation | Random pattern reassign (Pa_RR), |
| | Score based reassign (Pa_SR), |
| | Two points pattern swap (Pa_2SW) |
| Pattern related: FI | Pattern reassign first improvement (Pa_FIR), |
| | Two points pattern swap (Pa_2SW) |
| Mixed: FI | Relocate with pattern (MRPa), |
| | Cross with pattern (MCPa) |

VRP or single route optimization). Pattern related operators reassign the patterns of $n$ customers; we consider $n = 1, 2, \ldots, 6$ in our experiments. Because of the structure of LLH parameter design, our LLH repository contains 70 to 110 LLHs, depending on the problem instance.

**Algorithm Frameworks.** To test LLH management strategies, we use three hyperheuristic frameworks, Fig. 1. The only difference between the first two frameworks is the strategy used to organize different types of LLHs (mutation and FI). Both the simple hyperheuristics and learning based hyperheuristics (Sect. 4) can be applied in frameworks 1 and 2. The third framework supports a VNS-based method (Sect. 4.3). Compared to framework 2, it replaces the second stage (a single selection) with an ILS over a subset of the LLH repository.



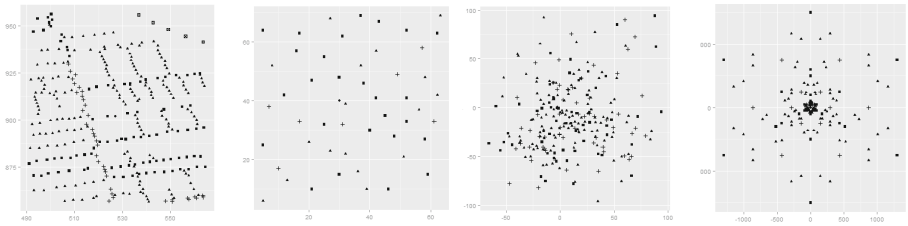

(a) Framework 1 (b) Framework 2 (c) Framework 3

**Fig. 1.** Hyperheuristic frameworks (the first two are modified from [23])

### 5.1 Problem Instance

Our data comprises 42 benchmark problems (summarised by [5]) and six instances from a real-world periodic maintenance problem[1]. We classify the problems according to their spatial characteristics (Fig. 2). Table 2 summarises each class. The six real-world instances are all street type. The big random benchmark problems have both a larger number of customers and greater clustering of data points than the small random class.

[1] The real-world data and associated best-performance results (Sect. 6) can be found at https://www-users.cs.york.ac.uk/~yujiec/.



(a) street style (real world example)

(b) Small random (from benchmark p02 [6])

(c) Big random (from benchmark pr05 [8])

(d) Symmetrical (From benchmark p32 [7])

**Fig. 2.** Examples of four types of spatial distributions in the PVRP instance set.

**Table 2.** PVRP instances. n is number of customers; m is number of vehicles; t is length of planning period in days. Benchmark labelling is from [5].

| Class | n | m | t | Average visit frequency | Number of problem instances |
|---|---|---|---|---|---|
| Street style | 240–324 | 3–5 | 6 | 1.6-2.1 | 6 |
| Small random | 50–100 | 1–6 | 2–10 | 1–2.1 | 10 (benchmark p01-p10) |
| Big random | 48–417 | 2–12 | 4–7 | 1.1–3 | 13 (benchmark p11-p13, pr01-pr10) |
| Symmetrical | 20–184 | 2–9 | 4–6 | 1.8-2 | 19 (benchmark p14-p32) |

## 6 Experimental Results and Analysis

### 6.1 Random Vs Learning Based Selection Strategies

A hyperheuristic needs an efficient selection strategy, because it is impractical to apply all LLHs exhaustively. The first experiment compares the SR selection strategy to the learning based strategies, RL, CF and BEBO. The experiment uses framework 1 (Fig. 1a) and OI acceptance. For RL and CF, we test using both the best 30 % and the best 80 % of LLHs in each iteration (See Sect. 4.2).

Each selection strategy is run 20 times on each instance of each of the four classes of problem, to give the percentage differences to the best-found benchmark route length of each instance. We then average the results for each class of problems.

The results in Table 3 show that, whilst acceptable, none of our solutions matches the best-found benchmark solution. Learning based selection strategies consistently out-perform SR LLH selection, with BEBO performing best. For both RL and CF, the limited CPU time makes it difficult for the hyperheuristics to produce competitive results for $w = 80$. In subsequent experiments we only use the best 30 % of LLHs.

**Table 3.** The average percentage difference to the best found solution over all instances in each group, for simple random (SR) and learning based hyperheuristics, using framework 1 and OI acceptance.

| | SR | RL(30 %) | CF(30 %) | RL(80 %) | CF(80 %) | BEBO |
|---|---|---|---|---|---|---|
| Street style | +8.90 | +5.36 | +6.08 | +6.47 | +6.64 | +4.90 |
| Small random | +4.67 | +2.12 | +2.28 | +2.15 | +2.33 | +2.06 |
| Big random | +4.32 | +4.14 | +4.32 | +4.28 | +4.35 | +4.13 |
| Symmetrical | +2.74 | +1.46 | +1.56 | +1.55 | +1.53 | +1.50 |

### 6.2 Impact of Algorithm Framework

Having shown that learning based selection strategies can manage a large number of LLHs in a simple hyperheuristic framework, we now consider the different hyperheuristic frameworks.

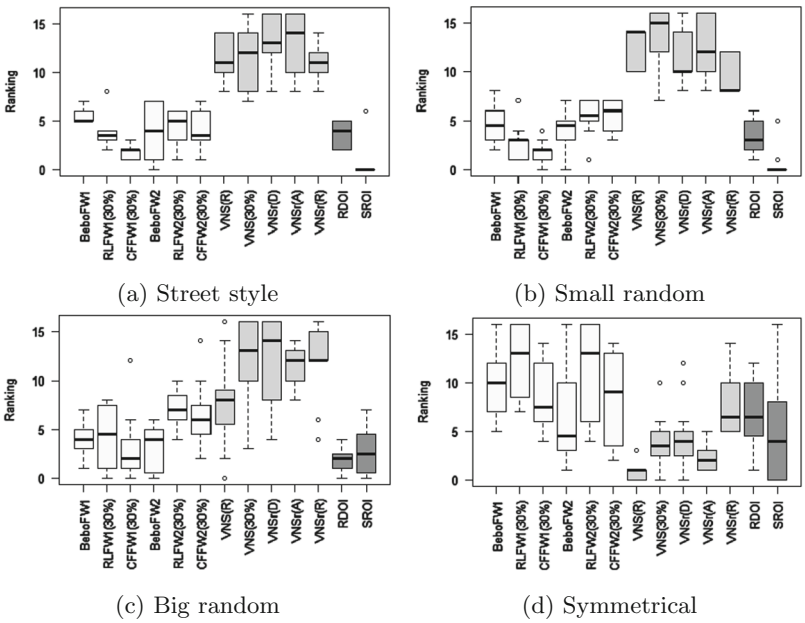
In framework 1, the OI acceptance rule means that mutation LLHs are unlikely to be favoured. In framework 2, a mutation operator is randomly selected, and is applied as long as it generates valid solutions, then FI LLHs are selected using a learning based strategy, as above. Framework 2 is similar to framework 1 when we use the all-move-accept rule, but, whereas framework 1 evaluates the mutation and FI operators together, framework 2 allows separate consideration.

The results in Table 4 show that framework 2 improves the performance of both RL and CF hyperheuristics for all types of problem instances, and, BEBO does not show obvious difference between framework 1 and 2.

Framework 3 uses the VNS-based algorithms; the main difference to framework 2 lies in the use of ILS once a FI operator is selected. Five variants are tested. The first two use GVNS (Algorithm 2), with VND using, respectively, randomly ordered FI LLHs (VNS(R)) and the best 30 % of FI LLHs (VNS(30 %)). Three variants use GVNSr (Algorithm 3), with random, ascending or descending FI LLH ordering determined using utility (respectively, VNSr(R), VNSr(A), VNSr(D)). We compare performance with the above three framework 1 and

**Table 4.** The average percentage differences to the best found solutions over all instances in each group, for learning based hyperheuristics using frameworks 1 and 2 (FW1, FW2)

| Instances | RL(30 %) | | CF(30 %) | | BEBO | |
|---|---|---|---|---|---|---|
| | FW1 | FW2 | FW1 | FW2 | FW1 | FW2 |
| Street style | +5.36 | **+5.26** | +6.08 | **+5.54** | **+4.90** | +5.31 |
| Small random | +2.19 | **+1.88** | +2.28 | **+1.90** | +2.06 | **+2.03** |
| Big random | +4.14 | **+3.93** | +4.32 | **+3.88** | +4.13 | **+4.12** |
| Symmetrical | +1.46 | **+1.45** | +1.56 | **+1.54** | **+1.50** | +1.56 |




(a) Street style (b) Small random

(c) Big random (d) Symmetrical

**Fig. 3.** Ranking of hyperheuristics for PVRPs. Higher rank is better.

three framework 2 strategies, plus random descent (RDOI) and simple random (SROI) embedded in framework 1. We then rank the performance of the 12 combinations of framework and LLH selection strategy, awarding 16 points to the best performing hyperheuristic, then $14, 12, 10, 8, 7, \ldots 1, 0$ points successively to worse performing hyperheuristics.

Figure 3 shows a small difference in performance between frameworks 1 and 2. Compared to the framework 3 results, they are both generally low-ranking for all cases except the symmetrical benchmark problems. This shows the positive impact of using ILS. Among framework 3, the five VNS-based algorithms show similar ranking, except for big random, where VNS(R) is not highly ranked; random selection of the shaking operator combined with random ordered FI LLHs (VNSr(R)) is the most robust over all classes of problem.

### 6.3 Scalability

The PVRP is NP-hard [15]. One of its biggest challenges is the rate of growth in complexity with problem size. In preliminary experiments, we determined that the performance of algorithms on symmetrical and non-symmetrical problems is very different. To test the scalability of our hyperheuristics, we first group the problem instances into symmetrical and non-symmetrical problems and then order them by the number of customers. Each method is runs 20 times.

Figure 4 shows that SROI has the worst scalability in both symmetrical and non-symmetrical problems. For the other algorithms, there is little difference in

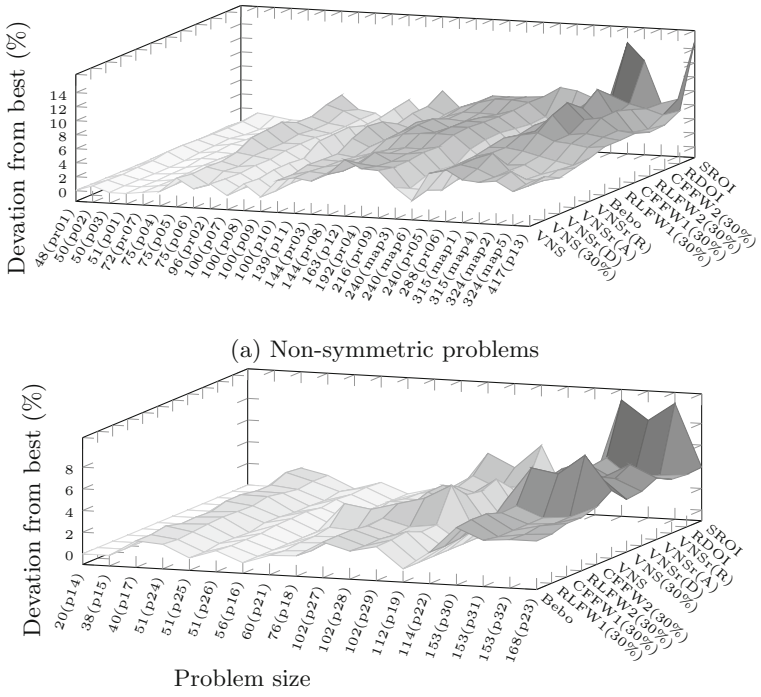


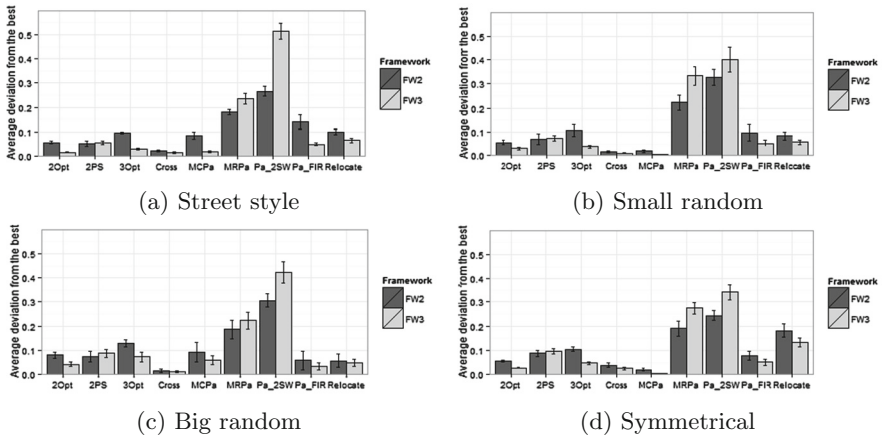

**Fig. 4.** Performance of hyperheuristics tested on PVRP with various sizes

performance on problems with fewer than about 60 customers. VNS-based algorithms are the most robust across non-symmetrical instances with 150 to 420 customers. However, performance decreases dramatically for VNS-based algorithms applied to bigger problem instances in the symmetrical data set.

### 6.4 LLH Usage Analysis

Whilst hyperheuristics need little specialised design, the LLH repository does need thought. In this experiment, we explore the usage of LLHs by the different hyperheuristics. We use frameworks 2 and 3, which manage the mutation and FI operators separately. The results focus on the 9 FI LLHs, since there is no learning in mutation operator selection.

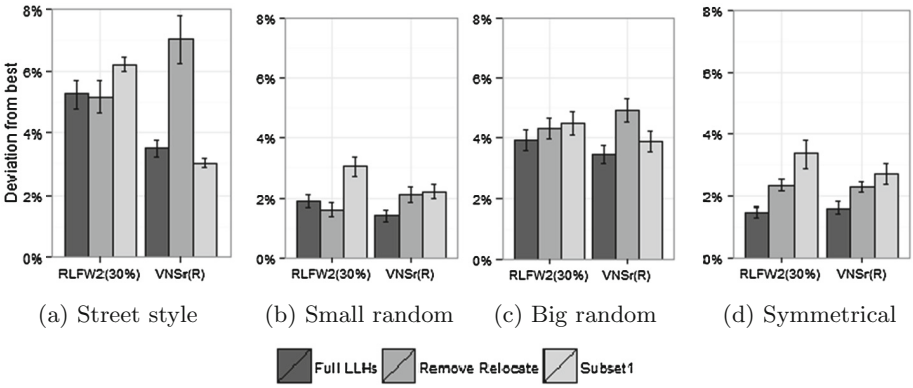
Figure 5 summarises average usage of FI LLHs for all learning based algorithms using framework 2 (BeboFW2, RLFW2(30 %), CFFW2(30 %)) and all VNS-based algorithms using framework 3 (VNS(R), VNS(30 %), VNSr(D), VNSr(A), VNDr(R)). The stronger LLHs are favoured more in framework 3 than in framework 2. "Relocate with pattern" (MRPa) and "two points pattern swap" (Pa_2SW) are the most applied LLHs by all hyperheuristics. Since we




(a) Street style (b) Small random

(c) Big random (d) Symmetrical

**Fig. 5.** FI LLHs usage for different types of problem. Results show the mean value of percentage of each LLH are applied during the search, where 0.1 stands for 10 %. Error bars show 95 % confidence interval.




(a) Street style (b) Small random (c) Big random (d) Symmetrical

**Fig. 6.** Performance of RLFW2(30 %) and VNSr(R) using different subset of LLHs. Error bars show 95 % confident interval. The subset1 removes the most used LLH (Pa_2SW) and all mutation operators except Pa_RR

are using an OI strategy, this implies that they consistently produce improved solutions.

The "Relocate" operator is preferred in symmetrical problems, but not in other instances. The importance of this operator is emphasised by the big reduction in performance when the "relocate" operator is removed (Fig. 6d).

To further explore the contribution of specific LLHs in improving PVRP solutions, we test the two best performing hyperheuristics for frameworks 2 and 3 (RLFW2(30 %) and VNSr(R)) with different subsets of the original LLHs. Each method is run over all problem instances; results are the average of 20 runs.

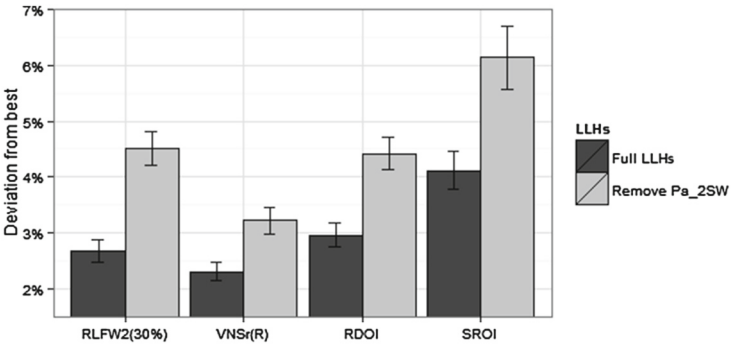



**Fig. 7.** Impact of removing Pa_2SW on four hyperheuristics over all problem instances. Error bars show 95 % confidence intervals.

Figure 6 shows a change in performance after removal of the most-used FI LLH (Pa_2SW) and all mutation operators except Pa_RR (Subset 1): the performance of RLFW2(30 %) and VNSr(R) decreases dramatically for the small random and symmetrical problems. However, there is little difference for the big random instances, and we even find a small improvement for VNSr(R) on street style problems. One interpretation of this result is that the strongly-performing FI LLHs, which are most effective in small and symmetric problems, tend to become stuck in local optima in the street style and big random problems. Further work is needed to understand why removing the "relocate" operator affects performance on street style problems more than less-structured spatial distributions.

To explore the robustness of different hyperheuristics when we remove the strongest LLH (Pa_2SW), we extend the LLH subset experiments to SROI and RDOI. VNSr(R) shows the best robustness (Fig. 7). Comparing the RLFW2(30 %) with VNSr(R) and SROI with RDOI, the algorithms with ILS mechanisms are more robust than the algorithms without ILS.

### 6.5 Comparison Between Hyperheuristics and Other Meta-Heuristics

This section compares the two best performing hyperheuristics from framework 2 and 3 (RLFW2(30 %) and VNSr(R)), to published meta-heuristics which have been designed or tailored for PVRP, including (parallel) tabu search [8,16], scatter search [14], VNS [5], record-to-record ILP [9] and hybrid Genetic Algorithm (GA) [15]. No comparative data exists for our street style data set.

Benchmark research uses 32 instances collected from early work on PVRP (the old data set). Cordeau [8] presents 10 additional PVRP instances (the new data set). We present our results for these two groups, because some research has not tested both groups. Table 5 reports the percentage difference in average performance from the best found (summarised in [15]) over these two data sets.

**Table 5.** Performance on PVRP benchmarks compared with meta-heuristics; tabu search (CGL) [8], scatter search (ALP)[14], VNS (HDH)[5], record-to-record ILP (GGW) [9], hybrid-GA (VCGLR)[15], parallel tabu search (CM) [16]

| | RLFW2(30 %) | | VNSr(R) | | CGL | ALP | HDH | GGW | VCGLR | CM |
|---|---|---|---|---|---|---|---|---|---|---|
| | Avg. 20 run | Avg. (best) | Avg. 20 run | Avg. (best) | 1 run | - | Avg. 10 run | - | Avg. 10 run | Avg. 10 run |
| Old data (%) | 1.86 | 1.08 | 1.77 | 0.93 | 1.8 | 1.57 | 1.6 | 1.11 | 0.032 | 0.044 |
| New data (%) | 3.88 | 2.40 | 3.44 | 2.12 | 2.82 | - | 1.86 | - | 0.071 | 0.091 |

Our hyperheuristics achieve competitive results compared to the tabu search [8], scatter search [14] and VNS [5] for the "old data" set. For the relatively larger "new data" set, we achieve close to the best found solutions in most cases. The hyperheuristic approaches are about 1 % worse than these problem-specific algorithms, in terms of total route distance.

Compared to the hybrid-GA, which out performs all the other algorithms, our hyperheuristics produce routes that are about 2 % longer on average. However, hyperheuristics do not require any knowledge directly from the solution space and require minimal design effort, whereas the meta-heuristics need to be designed and tailored for each problem.

## 7 Conclusion

Our analysis of hyperheuristics for PVRP shows that both learning selection strategy and ILS have positive impacts on an algorithm's performance and enhance the scalability. ILS also improves the robustness of hyperheuristics when a poor LLH set is given, because ILS concentrates on a neighbourhood structure until it reaches a local optimum, whilst approaches without ILS have a wider, but shallower, exploration within the search space.

Our hyperheuristics find solutions that are almost as good as those published for meta-heuristics. Since all experiments have limited CPU time, it is possible that this is due to the hyperheuristics' additional overhead in applying search at the LLH selection level. The hyperheuristics are more adaptable to new problems: our results show that hyperheuristics can efficiently manage a large LLH set and automatically select appropriate LLHs.

The tested hyperheuristics show similar performance on real-world street style problem instances and random instances, but the symmetrical benchmarks tend to favour different strategies and LLHs. This suggests that symmetrical instances are not a good indicator of algorithm performance for real-world PVRP.

"Relocate with pattern" and "two points pattern swap" are the most applied LLHs across all PVRP hyperheuristics: these LLHs make most improvements during the search. However, experiments on LLH subsets show that a strong LLH may lead to premature local optima; further work is needed on the effect of structure in real-world problems, and on ways to measure "strong" LLHs.

For PVRP, we show that hyperheuristics perform similarly to problem-specific meta-heuristics, despite their working mechanism potentially increasing the complexity of solving a specific problem within limited time. We are working on improving hyperheuristic efficiency, and investigating whether the positive impact of learning based selection and ILS translates to other problem domains.

**Acknowledgement.** The authors would like to thank Gaist Solutions Ltd. for providing data. This research is part of the LSCITS project funded by the EPSRC.

## References

1. Cowling, P.I., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)
2. Bilgin, B., Özcan, E., Korkmaz, E.: An experimental study on hyper-heuristics and exam timetabling. In: Burke, E.K., Rudová, H. (eds.) PATAT 2007. LNCS, vol. 3867, pp. 394–412. Springer, Heidelberg (2007)
3. Remde, S., Cowling, P.I., Dahal, K., Colledge, N., Selensky, E.: An empirical study of hyperheuristics for managing very large sets of low level heuristics. J. Oper. Res. Soc. **63**(3), 392–405 (2011)
4. Kalender, M., Kheiri, A., Özcan, E., Burke, E.K.: A greedy gradient-simulated annealing selection hyper-heuristic. Soft Comput. **17**(12), 2279–2292 (2013)
5. Hemmelmayr, V.C., Doerner, K.F., Hartl, R.F.: A variable neighborhood search heuristic for periodic routing problems. Eur. J. Oper. Res. **195**(3), 791–802 (2009)
6. Christofides, N., Beasley, J.E.: The period routing problem. Networks **14**(2), 237–256 (1984)
7. Chao, I.M., Golden, B.L., Wasil, E.: An improved heuristic for the period vehicle routing problem. Networks **26**(6), 25–44 (1995)
8. Cordeau, J.F., Gendreau, M., Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks **30**(2), 105–119 (1997)
9. Gulczynski, D., Golden, B., Wasil, E.: The period vehicle routing problem: new heuristics and real-world variants. Transp. Res. Part E: Logistics Transp. Rev. **47**(5), 648–668 (2011)
10. Beltrami, E., Bodin, L.: Networks and vehicle routing for municipal waste collection. Networks **4**(1), 65–94 (1974)
11. Russell, R., Igo, W.: An assignment routing problem. Networks **9**(1), 1–17 (1979)
12. Tan, C.C.R., Beasley, J.E.: A heuristic algorithm for the period vehicle routing problem. J. Omega **12**(5), 497–504 (1984)
13. Russell, R.A., Gribbin, D.: A multiphase approach to the period routing problem. Networks **21**(7), 747–765 (1991)
14. Alegre, J., Laguna, M., Pacheco, J.: Optimizing the periodic pick-up of raw materials for a manufacturer of auto parts. Eur. J. Oper. Res. **179**(3), 736–746 (2007)
15. Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W.: A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper. Res. **60**(3), 611–624 (2012)
16. Cordeau, J.F., Maischberger, M.: A parallel iterated tabu search heuristic for vehicle routing problems. Comput. Oper. Res. **39**(9), 2033–2050 (2012)
17. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Oper. Res. **12**, 568–582 (1964)

18. Gendreau, M., Hertz, A., Laporte, G.: New insertion and post optimization procedures for the traveling salesman problem. Oper. Res. **40**(6), 1086–1095 (1992)
19. Groër, C., Golden, B., Wasil, E.: A library of local search heuristics for the vehicle routing problem. Math. Program. Comput. **2**(2), 79–101 (2010)
20. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. Metaheuristics: Computer Decision-Making, pp. 523–544. Springer, New York (2004)
21. Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**(3), 449–467 (2001)
22. Hansen, P., Mladenović, N., Moreno Pérez, J.A.: Variable neighbourhood search: methods and applications. Ann. Oper. Res. **175**(1), 367–407 (2010)
23. Özcan, E., Bilgin, B., Korkmaz, E.: A comprehensive analysis of hyper-heuristics. Intell. Data Anal. **12**(1), 3–23 (2008)