A Combined Generative and Selective Hyper-heuristic for the Vehicle Routing Problem

Kevin Sim
School of Computing
Edinburgh Napier University
Merchiston Campus
Edinburgh, EH10 5DT
k.sim@napier.ac.uk

Emma Hart
School of Computing
Edinburgh Napier University
Merchiston Campus
Edinburgh, EH10 5DT
e.hart@napier.ac.uk

ABSTRACT

Hyper-heuristic methods for solving vehicle routing problems (VRP) have proved promising on a range of data. The vast majority of approaches apply *selective* hyper-heuristic methods that iteratively choose appropriate heuristics from a fixed set of pre-defined low-level heuristics to either build or perturb a candidate solution. We propose a novel hyper-heuristic called GP-MHH that operates in two stages. The first stage uses a novel Genetic Programming (GP) approach to evolve *new* high quality constructive heuristics; these can be used with any existing method that relies on a candidate solution(s) as its starting point. In the second stage, a perturbative hyper-heuristic is applied to candidate solutions created from the new heuristics. The new constructive heuristics are shown to outperform existing low-level heuristics. When combined with a naïve perturbative hyper-heuristic they provide results which are both competitive with known optimal values and outperform a recent method that also designs new heuristics on some standard benchmarks. Finally, we provide results on a set of rich VRPs, showing the generality of the approach.

Keywords

Hyper-heuristic, VRP, Genetic Programming

1. INTRODUCTION

The Vehicle Routing Problem (VRP) is a well known optimisation problem, of considerable practical importance within many industries. It exists in many forms, defined for instance by characteristics of vehicles, the requirements of the customers they service, or the type and the availability of goods at a depot. Many sophisticated meta-heuristic methods exist for tackling these difficult combinatorial optimisation problems; while they often find high quality solutions, they tend to be specialised to specific variants of VRP.

GECCO '16, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908812.2908942

In recent years, attention has turned to hyper-heuristic approaches which aim to reduce the time required for algorithm design by providing a general framework that controls application of lower level heuristics. Although trading algorithm-design time against quality, many promising results have been reported, e.g. [3].

Burke *et al.* [3] describe a classification of hyper-heuristics which separates methodologies with respect to two dimensions; the nature of the heuristic' search space, and the sources of feedback information. With respect to the heuristic search-space, two methodologies are apparent. *Selection* methods choose or select from existing heuristics while *Generation* methods generate completely new heuristics. Both methods can be sub-divided into *perturbative* and *constructive* heuristic methods. Perturbative methods iteratively modify an initial candidate solution (in single-point hyper-heuristic) or solutions (in multi-point hyper-heuristic), while constructive methods consider partial (or empty) candidate solutions, and iteratively extend them [4].

With respect to vehicle routing, the most common hyper-heuristic approaches fall within the *selection* category, e.g. [9, 16, 21]. In the majority of approaches, a single feasible candidate solution is constructed according to a fixed method, which is then operated on by the hyper-heuristic. No attention is paid to the quality of the initial candidate solution, which may limit the overall performance of the heuristic. Improving on this, *constructive* methods such as [9, 10] search the space of sequences of constructive *and* perturbative pairs of low-level heuristics. These sequences are applied in order to both construct and improve partial solutions. However, a potentially limiting factor here is the availability of a diverse set of constructive heuristics; for example, Garrido [10] uses a set of only four constructive heuristics; in contrast, many more perturbative heuristics exist [3]. Mlejnek *et al.* use the same set of four constructive heuristics in an evolutionary hyper-heuristic called HyperPOEMS [14] that also combines selection of constructive and perturbative methods. Addressing this weakness, Drake *et al.* [8] propose a hyper-heuristic methodology that uses Grammatical Evolution to simultaneously evolve novel constructive and perturbative heuristics. Their proposed hyper-heuristic initialises a single candidate solution using the construction heuristic, that is then iteratively operated on by the new perturbation heuristics.

The contribution of this paper is twofold: First, the introduction and evaluation of a novel method for generating new constructive heuristics. The method addresses the two concerns raised above: the lack of available constructive methods and the potentially limiting quality of constructing a single weak candidate solution. The novel construction heuristics can be used in a purely constructive hyper-heuristic methodology or with *any* hyper-heuristic or meta-heuristic method that requires the creation of candidate solutions. Secondly, we propose a new multi-point hyper-heuristic method called *GP-MHH* that uses a two-stage process. In the first phase, genetic programming (GP) is used to evolve a population of construction heuristics. In the second phase, the evolved heuristics are used to construct a population of candidate solutions. A perturbative-selection hyper-heuristic, given the name Memetic Hyper-heuristic (MHH), is then applied. We show that even using a naïve perturbative strategy adapted from [21], the method outperforms the recent grammatical evolution approach of Drake *et al.* [8].

We evaluate our approach using two sets of problems: the Solomon benchmark instances [20] and a new set of rich VRP problems described in [19]. Results are evaluated according to the following criteria: performance; relation to optimality (where known); comparison to published methods. We compare the GP stage of GP-MHH to results obtained on the Solomon instances using 7 standard constructive heuristics and compare the complete procedure to the recent Grammatical Evolution approach from Drake [8] and to the optimal or best known solutions. To our knowledge the encouraging results presented here for the rich VRP instances constitute the first attempt to improve upon the solutions presented in [19] and illustrate the generality of the approach on highly constrained instances.

2. RELATED WORK

This section covers relevant background work relating to hyper-heuristic approaches to Vehicle Routing Problems.

2.1 Perturbative-selection hyper-heuristics

These methodologies aim to improve a candidate solution or solutions (in single-point and multi-point hyper-heuristic respectively) through a process of automatically selecting and applying a heuristic, choosing from a set of pre-defined low-level strategies. Several applications of this method to VRP are apparent in the literature.

Pisinger and Ropke [16] extend a large neighbourhood search framework first presented in Shaw (1998) with an adaptive layer that selects from a set of insertion and removal heuristics to modify a solution. This layer either intensifies or diversifies the search, according to scores for each heuristic that are accumulated over the course of a run. Seven *ruin* and two *recreate* heuristics are available for selection. Tests on standard benchmarks from the literature covering five variants of the vehicle routing problem showed the method was highly promising, improving on the best known solutions for some instances.

Meignan *et al.* [12] describe a self-adaptive and distributed approach that combines agents and hyper-heuristics. A group of agents concurrently explore the search space of a given problem instance. Each agent modifies a solution with a set of operators. The selection of these operators is determined by heuristic rules, dynamically adapted by individual and collective learning mechanisms. Evaluation on fourteen instances of the Christofides benchmark [6] showed that the approach was competitive with some other algorithms in terms of speed and quality. Misir *et al.* [13] address a real-world problem relating to concrete delivery. They propose a new move acceptance method that determines whether a solution just modified by a heuristic should be accepted, based on its quality and the current state of the search. Their method adaptively sets the threshold value based on previous history.

Walker *al* [21] describe a multi-point approach in which a population of candidate solutions is first constructed stochastically. Both crossover and perturbation operators are iteratively applied to improve the population. Heuristics are selected at random in their baseline approach, or using an online learning mechanism in an adaptive version.

2.2 Constructive-selective hyper-heuristics

In contrast to perturbative methods which start from a candidate solution, *Constructive-selective* methodologies aim to build a solution incrementally. Typically, a hyper-heuristic methodology will iteratively select from a set of available constructive heuristics until a complete solution is created.

Garrido and Castro [9] use a hill-climbing-based hyper-heuristic to solve the capacitated vehicle routing problem (CVRP) that selects from both constructive and perturbative heuristics. Their hyper-heuristic manages a sequence of constructive-perturbative pairs of low-level heuristics which are applied sequentially in order to construct and improve partial solutions. The low-level heuristic set includes four constructive heuristics from the literature, and six well-known perturbative heuristics. The approach was shown to be competitive on the set of benchmark Christofides instances [6]. A further paper from same authors describes an evolutionary hyper-heuristic for solving the dynamic vehicle routing problem (DVRP) [10]. As in their previous work, a sequence of heuristics is evolved, now extended to include three types of low-level heuristics: constructive, perturbative, and noise heuristics. The approach provided competitive results when compared against well- known methods from the literature.

2.3 Heuristic generation methodologies

Generative hyper-heuristics are a relatively recent development within the field. These approaches search a space of heuristics constructed from component parts, rather than a space of pre-defined heuristics. They therefore generate new heuristics that can be later be reused on new problem instances. The method has proved particularly successful in both the scheduling domain [2] and packing domains [5]. However, to the best of our knowledge, the only example of the use of a generative method within VRP is recent work by Drake *et al.* [8] that uses grammatical evolution (GE) as a tool to evolve the components of a variable neighbourhood search (VNS) framework. The framework generates a new constructive heuristic to create a single initial solution, along with new neighbourhood move operators to then perturb the solution. The framework is tested on a number of benchmark instances, showing promising results on benchmark Capacitated Vehicle Routing Problems (CRVP) from Augerat [1] but less well on routing problems from Solomon [20] that contain time windows (VRPTW).

Conceptually, our approach is similar to that of Drake *et al.* [8] in that we aim to generate novel constructive heuristics from component parts. However, it differs in that we focus on evolving a *population* of new constructive heuristics, using GP rather than GE. The population of heuristics is then used to construct a population of candidate solutions. These can then be passed to any selective hyper-heuristic (or meta-heuristic) that attempts to refine the solution(s).

3. IMPLEMENTATION

The new hyper-heuristic, named GP-MHH, is shown in Algorithm 1. The hyper-heuristic defines two separate stages. In Stage 1, a novel Genetic Programming (GP) approach is used to automate the design of constructive heuristics for the VRP. The population of heuristics generated are then used to create a set of candidate solution(s) that are used to seed the population of a perturbative hyper-heuristic algorithm. Note that given a population of size p, then $P \geq p$ candidate solutions can be generated, given that random breaking of ties when applying a heuristic introduces a stochastic element. In Stage 2, we illustrate the approach using a simple hyper-heuristic adopted from [21] as the perturbative hyper-heuristic.

Algorithm 1 Two-stage hyper-heuristic GP-MHH

1: $t \leftarrow$ total function evaluations
2: Apply Genetic Programming algorithm GP(population size $= p$, $x < t$ evaluations)
3: $C \leftarrow$ final population from GP
4: Initialise $P \geq 1$ candidate solution(s) using set of C heuristics
5: Apply Perturbative Hyper-Heuristic($(t-x)$ evaluations)

Given the two stage process, it is necessary to divide the evaluation budget between the two stages. In this work, we used a fixed division for all instances in which the majority of the budget is devoted to Stage 2. This is based on preliminary experiments which suggested that the GP runs stagnate quickly. We note however that in future, there is much scope for using an adaptive approach which automatically detects a lack of improvement in stage 1 and switches to stage 2.

3.1 Constructive Hyper-heuristic

Genetic Programming is used to evolve a function that determines a priority for each operation that can potentially be inserted into the schedule. A construction heuristic is composed of a *route selector* and a GP tree — the route selector is assigned to each tree on initialisation, chosen from the list in Table 1. A heuristic iteratively builds a solution. Each iteration the heuristic's route selector chooses an open route from those available and any jobs remaining to be scheduled that meet the constraints of the associated vehicle are considered for insertion. Each suitable job is checked at each position on the route. If all jobs on the route satisfy their time constraints the job is added to a list of potential jobs. The list is then ranked using the GP tree based upon characteristics of the job under consideration and / or the previous job visited. The job and position that emerges as the winner is chosen for scheduling. In the case of a tie the job and position to be scheduled next is chosen randomly from the list of winners. The GP algorithm is given in Algorithm 2.

This is a conventional tree based algorithm: ramped half and half is used for initialisation, and crossover and mutation use the common subtree methods described in [11].

Algorithm 2 GP Pseudo Code

Require: $\mathcal{P}opulation = \emptyset$
Require: $popSize$, $p_{crossover}$, $p_{mutation}$, $maxIterations$, $maxInitialDepth$, $maxDepth$
while $|\mathcal{P}opulation| < popSize$ **do**
  $\mathcal{P}opulation = \mathcal{P}opulation + initialiseRandomHeuristic()$
  $evaluate(\mathcal{P}opulation)$
end while
repeat
  $\mathcal{N}ewPopulation = \emptyset$
  while $|\mathcal{N}ewPopulation| < popSize$ **do**
    $parent1 = rouletteSelection(\mathcal{P}opulation)$
    $parent2 = rouletteSelection(\mathcal{P}opulation)$
    if $U[0,1) < p_{crossover}$ **then**
      $child = crossover(parent1, parent2)$
    else
      $child = parent1$
    end if
    if $U[0,1) < p_{mutation}$ **then**
      $child = mutate(parent1, parent2)$
    end if
    $evaluate(child)$
    $\mathcal{N}ewPopulation = \mathcal{N}ewPopulation \cup child$
  end while
  $\mathcal{N}ewPopulation = \mathcal{N}ewPopulation \cup \mathcal{P}opulation$
  while $|\mathcal{N}ewPopulation| < popSize$ **do**
    $removeWorst(\mathcal{N}ewPopulation)$
  end while
until *stopping criteria met*

for stage 1 $popSize = 30$ $p_{crossover} = 0.1$ $p_{mutation} = 0.9$ $maxIterations = 10$ $maxInitialDepth = 7$ $maxDepth = 17$

Table 1: Route Selectors

Name	Description
FCFS	First Route Available
FPJ	Route with fewest possible jobs
FVFS	Route using the first Instantiated Vehicle
LCFS	Last Route Available
LTR	Least used route by time
MPJ	Route with most possible jobs
MSJ	Route with most scheduled jobs
MTR	Most used route by time
RR	Random Route

The set of terminal nodes implemented are described in Table 2. A wide variety of terminals are used so that the procedure generalises across a range of VRP types, e.g. those with/without time windows etc. For the Solomon instances the nodes designed to return temporal information relating to travel times between customers simply return *distance* as specified in [20]. Although the inclusion of these nodes increases the search space, it has a significant benefit in enabling the method to be used across a range of VRP classes (e.g VRPTW, CVRP etc.) [19]. The function nodes used are $+, -, \times, \div$ (protected divide returns ∞), *compare*(compares 2 child nodes a, b and returns $-1, 0, 1$ for $a < b$, $a = b$, $a > b$ respectively). IGTZ – (If child evaluates as greater than zero then evaluate child 2 else evaluate child 3), Max–Returns maximum of two child nodes.

Table 2: Terminal Nodes

Node	Description
ADJ	Average distance of proposed job to remaining jobs
DEM	Demand
DCP	Distance from current position
DD	Distance from depot
CWDS	Clarke Wright distance saving
DCPJ	Avg Distance of current position to remaining jobs
FCFS	First Come First Served
int	Random Integer Value
SLDTW	Slack in TW (start of time window - time from depot)
MCDC	Most constrained demand / capacity
rand	Random
SdDist	Standard deviation of distance to remaining jobs
SL	Slack in TW (end of tw - unloading start)
SWD	Sweep angle from depot position
SWC	Sweep angle from current position
TCP	Time from current position
TD	Time from depot
CWTS	Clarke Wright Time Saving
STW	Start of time window
W	Wait until Unloading Starts (start of TW - arrival)

3.2 Perturbative Hyper-heuristic

The perturbative hyper-heuristic that is investigated, described by Algorithm 3, is inspired by the naïve Iterated Local Search (ILS) algorithm described in [21] and built on top of the HyFlex framework [15]. We do not make use of HyFlex here, but simply incorporate a modified version of the algorithm into GP-MHH. The ILS hyper-heuristic described in [21] uses a set of mutation operators and a set of local search operators that are iteratively applied to improve on a single solution. In contrast the approach described here is implemented as a population based search incorporating two crossover operators as well as the same set of mutation and local search operators used in [21]. This allows Algorithm 3 to take advantage of the diverse set of solutions produced by the constructive heuristics evolved during stage 1. Additionally the constructive heuristics from stage 1 are used to repair infeasible solutions created during the application of evolutionary operators.

The ILS algorithm described in [21] has a variety of different ruin-create, mutation and local search operators at its disposal. In the system described here there is no distinction made between ruin-create operators and mutation operators. When selecting a *mutation* operator, the algorithm selects randomly from the set of all mutation and all ruin-create operators available. In addition to being used to initialise the population of the HH, the constructive heuristics evolved in the first stage are also incorporated into some of the evolutionary operators. When required to recreate or repair a partial solution, the ruin-recreate heuristics and the crossover heuristics randomly select a constructive heuristic taken from the final population evolved in stage 1. The evolutionary operators available to the HH are listed below. A fixed value of $\alpha = 10$ is used as the *search depth* for all local search and ruin-create heuristics.

Algorithm 3 MHH Pseudo Code

Require: $ConstructiveHeuristicsfromStage1 \neq \emptyset$
Require: $\mathcal{P}opulation = \emptyset$
Require: $popSize$, $p_{crossover}$, $p_{mutation}$, $p_{localSearch}$, $maxIterations$
while $|\mathcal{P}opulation| < popSize$ **do**
  $constructiveHeuristic = U[\mathcal{C}onstructiveHeuristics]$
  $\mathcal{P}opulation = \mathcal{P}opulation + constructiveHeuristic\#Solution$
end while
$evaluate(\mathcal{P}opulation)$
repeat
  $parent1 = tournamentSelection(\mathcal{P}opulation)$
  $parent2 = tournamentSelection(\mathcal{P}opulation)$
  if $U[0,1) < p_{crossover}$ **then**
    $child = crossover(parent1, parent2)$
  else
    $child = parent1$
  end if
  if $U[0,1) < p_{mutation}$ **then**
    $mutationHeuristic = U[MutationHeuristics]$
    $child = mutate(child)$
  end if
  if $U[0,1) < p_{localSearch}$ **then**
    $localSearchHeuristic = U[LocalSearchHeuristics]$
    $child = conductLocalSearch(child)$
  end if
  $evaluate(child)$
  $\mathcal{P}opulation = replaceWorst\mathcal{P}opulation, child$
until *stopping criteria met*

for stage 2 $popSize = 40$ $p_{crossover} = 0.2$ $p_{mutation} = 0.8$ $p_{localSearch} = 0.8$ $maxIterations = 9700$

Mutation / Ruin Create Operators
–*Two-opt*: Swap the position of two consecutive customers on a route.
–*Or-opt*: Moves two consecutive customers on a route to a different place.
–*Shift*: Moves a customer to another suitable route.
–*Interchange*: Swaps two customers from different compatible routes.
–*Time-based radial ruin*: Removes $U[1, \alpha]\%$ of customers from the solution based upon the proximity of their time window to a randomly chosen time within the duration of the complete schedule.
–*Location-based radial ruin*: Removes $U[1, \alpha]\%$ of customers from the solution, based upon the proximity of their location to a randomly chosen customer.

Local Search Operators
These heuristics are applied α times with equal or improved solutions retained after each application.
–*Shift*: Moves a customer to a different route.
–*Interchange*. Swaps two customers from different routes.
–*Two-opt*: Swaps the end sections of two routes.
–*GENI*: A customer is moved between two adjacent customers on a different route that are closest to it.

Crossover Operators
–*Combine*: Using two parent solutions, U[25%, 75%] of routes are copied from one parent to create a single offspring. Any non-conflicting routes from the second parent are added and the remaining customers are inserted using a randomly selected constructive heuristic from the first stage.
–*Longest Combine* All routes from two parent solutions are considered in descending order of the number of customers served. Any routes without duplicate customers that respect all of the problem constraints are added to create an offspring. Remaining customers are reinserted as before.

4. EXPERIMENTS

Two datasets are used for evaluation. The first is the Solomon benchmark set [20] that contains 56 instances of Capacitated Routing Problems with time-windows, divided into 6 classes, reflecting the layout of the customers and the tightness of the time constraints. The total summed route distance is used for fitness with no vehicle penalties.

The second set is a new set of Rich VRP instances that was recently described in[19] and can be accessed from [18]. These new instances incorporate many real-world features and constraints including heterogeneous, multi-compartment capacitated vehicle fleets, customer access and driver restrictions and include realistic travel times and travel distances derived from a real road network. We use a subset of 120 instances numbered 1001200 through 1001319 that use customers with locations correlated to population density surrounding a single urban depot extending in all directions into rural areas up to 240 minutes round trip travel time from the depot. The 120 instances are split into sets of 10 using 12 parameter combinations relating to number of vehicles (4, 8,16,32) and duration of time windows (60,120,240 minutes). Here, we use the authors suggested objective fitness metric which takes into account fixed and variable vehicle costs, penalties for broken constraints and outsourcing costs related to undelivered jobs. For details the reader is directed to the original publication. Unfortunately optimal solutions values are not known — however, the authors do provide a baseline result calculated from the method by which the problems were generated, which we compare to. We investigate the following questions:

- Do automatically generated *construction* heuristics compete with human-designed equivalents across a range of benchmark instances?
- Given a fixed number of evaluations, does dividing the budget between constructive and perturbative stages outperform a purely constructive hyper-heuristic?
- To what extent does GP-MHH compare to the known optimal/baseline solutions?
- To what extent does GP-MHH compare to existing hyper-heuristic methods?

A natural addition would be the comparison of GP-MHH to a purely perturbative approach, i.e. running the MHH algorithm from a randomly initialised population. Initial experiments indicated that this resulted in very poor results (partly as a result of the relatively small computational budget allowed) and hence we have not included these experiments in the paper. Future comparisons to a perturbative hyper-heuristic that uses an informed heuristic to construct an initial population would be instructive.

All reported results are from 30 repeated runs of the system. Algorithm parameters are as described above. A *total* function-evaluation budget of 10,000 evaluations is used in all experiments. For experiments that use only GP (i.e. no Stage 2), the population is of size of 500 and evolves for 20 generations. In the two-stage experiments, stage 1 is allocated only 300 evaluations, with the GP using a population of 30 for 10 generations. This parameter was chosen based on preliminary experimentation that indicated that following initial rapid improvement, GP makes slow progress. Results presented in the following section back up this observation. All other parameters for the GP and MHH algorithms are specified in Algorithms 2 and 3.

5. RESULTS

Hypothesis H1: *the newly evolved constructive heuristics produce higher quality solutions than existing construction heuristics.*

To investigate the quality of the newly evolved constructive heuristics as stand-alone heuristics, we compare results to 7 construction heuristics from the literature [20] and to the known optimal on the Solomon set. Three of these heuristics (SAV, S, I1) were used by [9, 10, 14] as low-level heuristics in selective hyper-heuristic methods. The RVRP dataset is not used in this test, as the constraints of these instances mean the construction heuristics available in the literature are not applicable without significant modification due to the complex constraints on vehicle types and customers.

Results are given in Table 3. Values in columns 2-8 are replicated directly from [20]. The penultimate column shows the best results obtained (and standard deviation) using heuristics evolved by the GP element of GP-MHH (i.e. without MHH) obtained from 30 runs and 10,000 function evaluations. Where there is a blank cell, no results were give in [20]. In each case, results are averaged over all instances in each of the 6 problem sets. The GP algorithm outperforms the human-designed heuristics on 5 out of 6 classes, only being narrowly beaten by Solomon's Insertion heuristic, I1, on class R1.

Hypothesis H2: *A two-phase approach split between construction and perturbation produces higher quality solutions that devoting the same number of evaluations to a purely constructive method*

Figure 1 considers the 56 Solomon instances separated into the standard classes (C1,C2, R1, R2, RC1, RC2). For a given class, in each of 30 runs, we sum the results of each instance across the class and divide by the summed known-optimal, hence providing a measure of *proximity* to optimal (as used in [8]). Figure 1 compares: (1) the result at the end of stage 1 (S1-GP), i.e. after 300 evaluations of GP; (2) the result obtained from 10,000 evaluations of GP only (S2-GP); (3) the result obtained from running GP-MHH (both stages) for 10,000 evaluations (S2-GP-MHH).

For each of the 6 sets, we observe (as expected) that S2-GP, i.e. GP run for 10,000 iterations outperforms S1-GP (300 iterations) although for set C2, the difference is less marked. GP-MHH outperforms both S2-GP and S1-GP on all sets. Shapiro-Wilk tests confirm that we cannot reject the null hypothesis that the data is normally distributed, hence we apply a t-test; for class C2 (S1-GP compared to S2-GP) and class RC2 (S2-GP compared to S2-GPMHH) the differences are significant at the 95% confidence level. For all other pairs, the p-values obtained are $<< 0.01$, hence results are strongly significant.

Table 3: Comparison between generated heuristics (GP) and 7 human heuristics.

	SAV	SWT	I1	I2	I3	NN	S	GP(s.d)	*Lit Opt*
C1	976.20	987.40	951.90	1049.80	1103.30	1171.20	940.80	**893.50(12.7)**	*828.38*
C2			692.70	921.50	1072.70	963.10	711.90	**603.54(9.5)**	*589.86*
R1	1498.90	1517.20	**1436.70**	1638.70	1651.70	1600.10	1499.70	1462.01(12.2)	*1210.34*
R2			1402.40	1470.70	1474.60	1472.30	1448.60	**1189.48(9.6)**	*951.03*
RC1			1596.50	1874.40	1849.70	1800.00	1804.50	**1580.07(8.7)**	*1384.16*
RC2			1682.10	1797.60	1816.40	1754.70	1735.70	**1446.81(30.4)**	*1119.24*

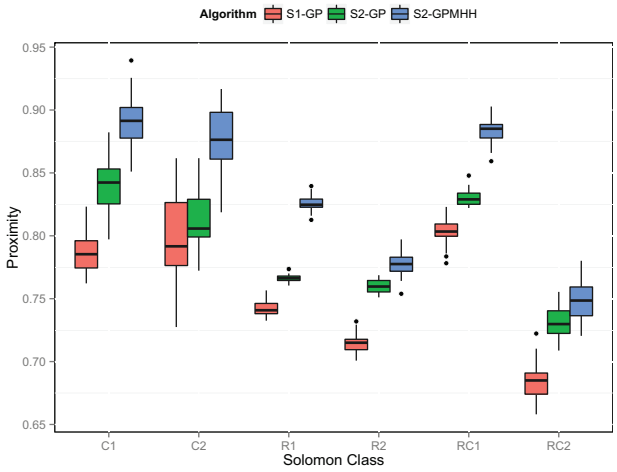

Figure 1: Solomon instances: comparison of GP-MHH at Stage-1, Stage-2 and GP only heuristic

For RVRP, as previously indicated, we compare results obtained with GP-MHH to the baseline published at [19]. Table 4 shows the results obtained, averaged over the 12 classes corresponding to different parameter combinations. In the first column V prefixes the number of vehicles and TW prefixes the duration of the time windows in minutes. Values for best, median and standard deviation are the result of 30 runs at the end of Stage-1 and the end of Stage-2. In each case, the obtained results is divided by the baseline result to obtain a proximity value. Note all results obtained by GP-MHH — including those from Stage-1 only (300 evaluations) — provide an improvement on the baseline (> 1.0). The Stage-1 results highlight the effectiveness of the construction heuristics. We reiterate the statement made earlier that the human-designed construction heuristics are not applicable to these instances, therefore the heuristics evolved using only Stage-1 can be considered useful. As with the Solomon instances, it is clear that considerable improvement is gained by running the two-stage GP-MHH.

Hypothesis H3:*Results are competitive w.r.t to known optimal/baseline solutions*

As indicated in the introduction section, hyper-heuristics cannot be expected to compare with specialised meta-heuristics. However, given the ubiquity of the Solomon instances in the VRP literature, it is informative to compare GP-MHH in more detail to the known optimal solutions. For each class of problems, we record the proximity of the best solution from the 30 runs on each of the n instances in each class. Figure 2 shows the distribution of results per class (i.e each box represent the n instances in the class). Note the for C1 and C2, a number of optimal instances are found. The median per class is within 90% of optimal for 3 classes, and within 80% for the remaining three. This suggests that GP-MHH is competitive, given only 10,000 function evaluations it finds high quality solutions very quickly.

For RVRP, we can only compare to the baseline values provided by [17] determined from the problem generation procedure. Figure 3 shows the results, split into 12 classes. We remark that some sets appear more difficult than others (e.g. S9- S12) and that the algorithm appears consistent: the inter-quartile range is small for each set. It is clearly possible to significantly improve on the baseline results. As this appears to be the first published results on this instance set it additionally provides a new benchmark for other researchers.

Table 4: Proximity to known results using GP-MHH on RVRP problem sets

	Stage 1 GP			Stage 2 MHH		
Problem Set	Best	Med	SD	Best	Med	SD
V4TW60	1.33	1.29	0.03	1.44	1.41	0.02
V4TW120	1.15	1.13	0.01	1.28	1.22	0.03
V4TW240	1.24	1.20	0.02	1.40	1.36	0.03
V8TW60	1.12	1.10	0.01	1.26	1.22	0.02
V8TW120	1.30	1.27	0.02	1.51	1.44	0.03
V8TW240	1.42	1.36	0.02	1.59	1.52	0.03
V16TW60	1.15	1.12	0.01	1.27	1.24	0.02
V16TW120	1.31	1.27	0.02	1.43	1.39	0.02
V16TW240	1.54	1.50	0.02	1.72	1.66	0.03
V32TW60	1.41	1.36	0.02	1.50	1.47	0.03
V32TW120	1.59	1.55	0.02	1.72	1.67	0.02
V32TW240	1.78	1.74	0.02	1.92	1.87	0.02

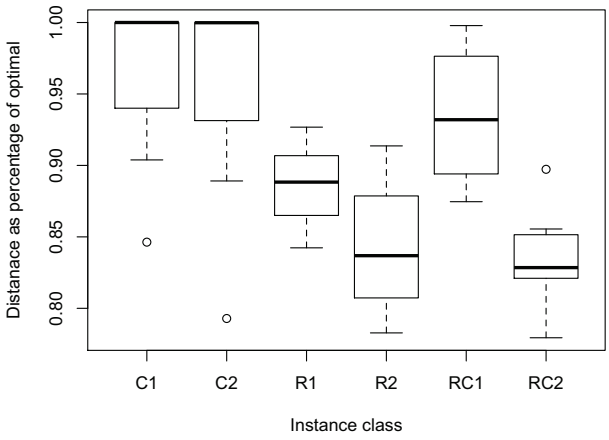

Figure 2: Distribution of best results per class for the Solomon problems, in relation to known optimal

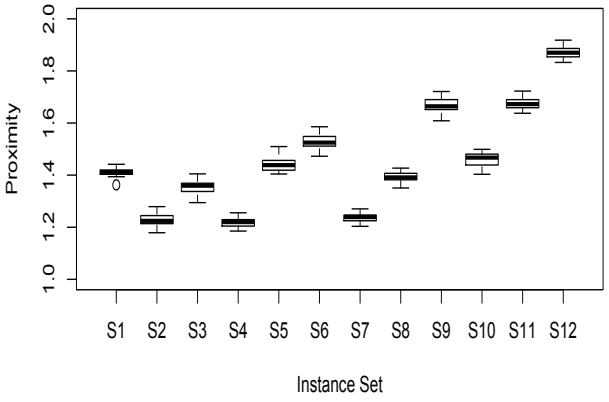

Figure 3: RVRP: comparison of best results obtained across each class to baseline solutions

Hypothesis H4: *Results are competitive with other hyper-heuristic methods that evolve constructive heuristics.*

To the best of our knowledge, the only other hyper-heuristic method that evolves novel construction heuristics is grammatical evolution hyper-heuristic from [8] called GE-PHH, who evaluate their method on 6 of the Solomon instances. Their results are obtained using a budget of 50,000 evaluations — 5 times more than the 10,000 used here. A comparison is given in Table 5 in which we compare the best result found using GP-MHH directly to the best result from [8]. GP-MHH outperforms GE-PHH method on 4 of the 6 instances indicating it is extremely competitive with similar methods.

Table 5: Comparison of GP-MHH with GE-PHH on 6 of Solomon's 100 customer instances

Instance	Optimal	GE-PHH / Proximity	GP-MHH / Proximity
C101	827.3	902.65 (0.92)	**828.94 (1.00)**
C102	827.3	1198.97 (0.69)	**917.12 (0.90)**
R101	1637.7	**1766.81 (0.93)**	1878.64 (0.87)
R102	1466.6	**1596.97 (0.92)**	1764.30 (0.83)
RC101	1619.8	1871.22 (0.87)	**1700.62 (0.95)**
RC102	1457.4	1771.46 (0.82)	**1618.04 (0.90)**

6. CONCLUSIONS

The paper describes a novel hyper-heuristic for solving vehicle routing problems. The new hyper-heuristic uses a two-stage process in which a fixed evaluation budget is split between constructive and perturbative stages. In contrast to the vast majority of hyper-heuristic approaches to VRP, we evolve *novel* construction heuristics during the first phase. We know of only one other method that achieves this (most simply sequence existing construction heuristics). Our hyper-heuristic differs from [8] in using standard *Genetic Programming* to evolve heuristics that prioritise a set of operations that can be scheduled. In contrast to [8] whose method evolves a single construction heuristic using grammatical evolution, we evolve a *population* of new construction heuristics. The population can be used with any single-point or multi-point perturbation hyper-heuristic by providing candidate solution(s) as a starting point. In addition, the population of evolved heuristics can be used to seed an initial population of a meta-heuristic method.

Experiments have shown that the evolved constructive heuristics are effective as stand-alone heuristics when compared to existing constructive heuristics. Furthermore, we have shown that by implementing them within a two-stage hyper-heuristic that follows up construction of a high-quality candidate solutions with an iterated local search phase, high quality results can be obtained. The *two-stage* hyper-heuristic is both competitive with existing methods on benchmarks, and shows significant improvement over a constructive only method over equivalent numbers of evaluations.

In addition, we have shown the applicability of the approach to real-world VRPSs through testing on a recently published set of data. This is a key contribution of the proposed method: existing constructive heuristics such as the well-known Clarke-Wright algorithm [7] cannot be applied to rich VRPs due to the complexity of the constraints encapsulated in the problems, e.g. relating to vehicle types and customer requirements. Significant modification of these heuristics would be required in order to construct candidate solutions that could be modified by a hyper-heuristic. This is a serious issue for the hyper-heuristic community if it wishes to move forward by showing its algorithms are applicable in the real-world. Not only does the new method provide a mechanism for constructing candidate solutions to these instances, it also provides the first benchmark data on this dataset for use by others researching in this area.

7. FUTURE WORK

A number of interesting potential directions exist for future work.

- We intend to consider an adaptive strategy that automatically determines when to switch between stages, by monitoring the progress of the stage 1 generative hyper-heuristic algorithm to determine when the algorithm is stagnating.
- Replacing the naïve hyper-heuristic strategy used in stage 2 with a more sophisticated perturbative approach is likely to yield further benefits
- Replacing the hyper-heuristic used in the second stage with a variety of readily available meta-heuristic algorithms would allow further analysis of the benefits of initialising the population using an ensemble of automatically generated constructive heuristics.
- Finally an interesting research direction is to evaluate the *reusability* of the heuristics, i.e. can a heuristic evolved for one instance construct good solutions when applied to a different instance.

8. ACKNOWLEDGMENT

The authors would like to acknowledge the funding provided under EPSRC Grant Number EP/J021628/1

9. REFERENCES

[1] P. Augerat, G. Rinaldi, J. Belenguer, E. enavent, A. Corberan, and D. Naddef. Computational results with a branch and cut code for the capacitated vehicle routing problem. *Tech. rep., RR 949-M, Universite Joseph Fourier, Grenoble*, 1995.

[2] J. Branke, S. Nguyen, C. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *Evolutionary Computation, IEEE Transactions on*, 20(1):110–124, Feb 2016.

[3] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics: a survey of the state of the art. *J Oper Res Soc*, 64(12):1695–1724, Dec 2013.

[4] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A classification of hyper-heuristic approaches. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer US, 2010.

[5] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. Automating the packing heuristic design process with genetic programming. *Evol. Comput.*, 20(1):63–89, Mar. 2012.

[6] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In *Combinatorial Optimization*, 1979.

[7] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Oper. Res.*, 12(4):568–581, Aug. 1964.

[8] J. H. Drake, N. Kililis, and E. Özcan. Generation of vns components with grammatical evolution for vehicle routing. In *Proceedings of the 16th European Conference on Genetic Programming*, EuroGP'13, pages 25–36, Berlin, Heidelberg, 2013. Springer-Verlag.

[9] P. Garrido and C. Castro. Stable solving of cvrps using hyperheuristics. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 255–262, New York, NY, USA, 2009. ACM.

[10] P. Garrido and M. Riff. Dvrp: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics*, 16:795–834, 2010.

[11] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[12] D. Meignan, J.-C. Creput, and A. Koukam. A coalition-based metaheuristic for the vehicle routing problem. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 1176 –1182, june 2008.

[13] M. Misir, W. Vancroonenburg, and G. Vanden Berghe. A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete. In *Proceedings of the 9th metaheuristic international conference (MIC'11)*, 2011.

[14] J. Mlejnek and J. Kubalik. Evolutionary hyperheuristic for capacitated vehicle routing problem. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '13 Companion, pages 219–220, New York, NY, USA, 2013. ACM.

[15] G. Ochoa, M. Hyde, T. Curtois, J. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, and E. Burke. HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. 2012.

[16] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8):2403–2435, 2007.

[17] Roll Project. Real world optimisation with life-long learning. *http://rollproject.org/*, 2015.

[18] K. Sim and E. Hart. Roll project rich vehicle routing benchmark problems 2015 available at http://dx.doi.org/10.17869/enu.2015.9367.

[19] K. Sim and E. Hart. A new rich vehicle routing problem model and benchmark resource. In *Proceedings of the The 11th edition of the International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems(EUROGEN-2015)*, 2015.

[20] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, 35(2):254–265, Apr. 1987.

[21] J. D. Walker, G. Ochoa, M. Gendreau, and E. K. Burke. Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In *Proceedings of the 6th International Conference on Learning and Intelligent Optimization*, LION'12, pages 265–276, Berlin, Heidelberg, 2012. Springer-Verlag.