An improved Genetic Programming Hyper-heuristic for the Uncertain Capacitated Arc Routing Problem

Jordan MacLachlan[1], Yi Mei[1], Juergen Branke[2], and Mengjie Zhang[1]

[1] Victoria University of Wellington, Kelburn 6140, NZ
{maclacjord,yi.mei,mengjie.zhang}@ecs.vuw.ac.nz
[2] The University of Warwick, Coventry CV4 7AL, UK
Juergen.Branke@wbs.ac.uk

Abstract. This paper uses a Genetic Programming Hyper-Heuristic (GPHH) to evolve routing policies for the Uncertain Capacitated Arc Routing Problem (UCARP). Given a UCARP instance, the GPHH evolves feasible solutions in the form of decision making policies which decide the next task to serve whenever a vehicle completes its current service. Existing GPHH approaches have two drawbacks. First, they tend to generate small routes by routing through the depot and refilling prior to the vehicle being fully loaded. This usually increases the total cost of the solution. Second, existing GPHH approaches cannot control the extra repair cost incurred by a route failure, which may result in higher total cost. To address these issues, this paper proposes a new GPHH algorithm with a new No-Early-Refill filter to prevent generating small routes, and a novel Flood Fill terminal to better handle route failures. Experimental studies show that the newly proposed GPHH algorithm significantly outperforms the existing GPHH approaches on the *Ugdb* and *Uval* benchmark datasets. Further analysis has verified the effectiveness of both the new filter and terminal.

Keywords: Arc Routing · Hyper-heuristic · Genetic Programming

1 Introduction

The Capacitated Arc Routing Problem (CARP) [9] is an important optimisation problem with many real-world applications such as city waste collection [1] and winter gritting [11, 10]. With the intention of accurately aligning CARP with reality, risk was introduced by [16] in presenting the Uncertain CARP (UCARP). At a high level, this consists of a set of vehicles which generate a number of routes (cycles) from a depot node, serving a number of edge-tasks at minimal cost subject to some constraints, e.g. the total demand of a route cannot exceed the vehicle's finite capacity. In UCARP, some information such as travel time and task demand is unknown, and can only be estimated prior to arrival at the edge in question.

In UCARP, preplanned solutions can fail and need to be adjusted in real time. For example the actual demand of a task can be greater than expected, and exceed the remaining capacity of the vehicle, or an edge along the planned path becomes impassable and the vehicle has to plan a detour. Traditional optimisation approaches (e.g. [23, 22]) that obtain a (robust) solution cannot handle this well, as they usually have high computational complexity, and cannot adjust the preplanned solution efficiently. *Routing policies* (e.g. [24, 14, 17]), on the other hand, are a promising strategy in making real-time decisions due to their low time complexity. A routing policy does not require any preplanned solution; instead, it models the UCARP as an online *decision making process*, where the routes are built over time by assigning the routes their next task at each step.

Manually designing effective routing policies is very time consuming, and requires a high level of domain expertise. To combat this, Genetic Programming Hyper-Heuristic (GPHH) can be applied to automatically evolve routing policies without the need of a human expert. The GP-evolved routing policy has shown great success in UCARP, and managed to achieve state-of-the-art solutions on many UCARP benchmark instances [14, 17].

However, the existing studies on GPHH for evolving routing policies are still preliminary, and most problem-specific characteristics have been neglected. As a result, the performance of existing GPHH methods are not satisfactory, having two main drawbacks. First, in the decision making process, a route tends to return to the depot early (e.g. when it still has sufficient capacity to serve more tasks), which generates many small cycles and leads to a large total cost. The existing methods are not intelligent enough to recognise and exclude this case. Second, the existing GPHH approaches cannot handle *route failure* (i.e. the actual demand of a task is larger than the remaining capacity) well, often leading to a large repair cost. This paper aims to propose new approaches to tackle the above two drawbacks, and develop an improved GPHH to evolve more effective routing policies. Specifically, the paper has the following research goals.

- Develop a *new decision making process* that explicitly prevents the routes from going back to the depot too early.
- Design a *new feature* to handle the route failure more effectively.
- Propose a new GPHH algorithm with the new decision making process and feature as a terminal.
- Examine the effectiveness of the newly proposed GPHH algorithm.

The rest of the paper is organised as follows. Section 2 introduces the background. Section 3 describes the newly proposed algorithm, including the new decision making process and feature. Section 4 conducts the experimental studies. Finally, the conclusions and future work are given in Section 5.

2 Background

2.1 Uncertain Capacitated Arc Routing Problem

In a UCARP instance, a graph $G(V, E)$ is given, and a set of vehicles with capacity Q are located at the depot $v_0 \in V$. Each edge $e \in E$ has (1) a positive *random* deadheading cost $dc(e)$, (2) non-negative *random* demand $d(e)$ and a non-negative *deterministic* serving cost $sc(e)$. If $d(e) > 0$, then e is called a *task*, and needs to be served. The goal is for the vehicles to serve all the tasks with the least total cost (sum of the serving cost plus deadheading cost of all edges in each route) subject to the following constraints:

– Each route starts and ends at the depot. Due to the route failures, the service of a task can be interrupted. A vehicle can therefore stop to replenish capacity early, before returning to complete the remaining service.
– Between two visits of the depot, the total demand served by the route cannot exceed the vehicle's capacity.

A *sampled* UCARP instance is a realised instance where each random variable has a sampled value. In a sampled UCARP instance, the sampled (actual) demand of a task is unknown until the vehicle completes its service. The actual deadheading cost of an edge is known exactly after the vehicle has traversed it. One can generate an arbitrary number of different sampled UCARP instances (e.g. using different random seeds) based on the same UCARP instance.

The objective of a UCARP instance is to find a solution (e.g. a predefined robust solution in proactive approaches or a routing policy in reactive approaches) that minimises the expected total cost across all the possible sampled UCARP instances based on that UCARP instance. In practice, it is impossible to enumerate all the possible sampled UCARP instances. Therefore, we will test our solution on a test set consisting of a large number of sampled UCARP instances.

2.2 Related Work

Solutions to static CARP, where costs of travel and task demand are known in full, range from exact mathematical methods on small instances [5], to tabu search methods [6, 12], memetic algorithms (MA) [13, 15] and Edge Based Histogram (EBH) methods [20]. A number of simple heuristics, such as Path-Scanning [13], Augment-Merge [9] and Ulusoy's single tour splitting method [21], have also been proposed to generate reasonably good solutions in a very short time. These heuristics can be used to generate initial solutions for the more advanced search algorithms.

To simulate the uncertain real world better, a variety of stochastic CARP models have been introduced, such as the CARP with stochastic demand [7], CARP with collaborating depots [19], and UCARP [16]. In [7], a genetic algorithm was proposed which took advantage of the concept of a 'slack' in determining the next task. In [4], a Branch-and-Price algorithm was proposed to consider the same stochastic task demand. UCARP was proposed in [16], considering four different stochastic factors to simulate the reality as closely as possible. There have been a number of studies for solving UCARP, including proactive approaches (e.g. [23, 22]) that optimise a robust solution, and reactive approaches that use Genetic Programming Hyper-Heuristics (GPHH) to evolve routing policies (e.g. [24, 14, 17]). Wang et al. proposed an Estimation of Distribution Algorithm with Stochastic Local Search (EDASLS) [22] that encompasses the work by [20]. They build routes for the static CARP instance using a path scanning heuristic [8] then construct an edge based histogram matrix on a template individual whenever a mutation operation occurs, per [20]. Further, they then perform a novel SLS method to manipulate the route string in an attempt to develop a better individual. To the best of our knowledge, EDASLS is the current state-of-the-art proactive approach in solving UCARP.

GPHH has achieved great success in solving dynamic combinatorial optimisation problems [3, 18, 2]. Based on the idea in [24], Liu et al. [14] proposed a GPHH to evolve routing policies for UCARP, and achieved promising results on the benchmark instances designed in [16]. In [14], a UCARP instance is modelled as a decision making process, where a routing policy is used for deciding the next task whenever a vehicle completes its current service. Then they use GP to evolve the routing policy. When route failure occurs, the vehicle simply returns to the depot in the middle of the service to refill, then returns to resume the interrupted service. When an edge failure occurs, the vehicle finds a detour using the updated graph topology. The work in [14] contains two contributions. First, it proposes a filter method to select a small set of candidate tasks at each decision point, improving the accuracy of the decisions made by the routing policy. Second, it designs a set of promising features to represent the current state, leading to better and more meaningful policies.

Mei et al. [17] extend the proactive approaches [24, 14] from a single-vehicle case to the general multi-vehicle case, developing a new meta-algorithm that generates routes simultaneously.

3 Proposed Algorithm

The standard framework of GPHH is described as follows.

1. Initialise a population of GP trees, each a routing policy (heuristic).
2. **Evaluate** the fitness of each GP tree using a training set.
3. Generate a new population by crossover/mutation/reproduction.
4. If stopping criteria is met, stop. Otherwise, go back to Step 2.

The evaluation of a GP tree on a training instance is essentially a decision making process with that tree as the routing policy. The fitness of the GP tree is set to the average total cost of the generated solutions on the training instances. A decision making process is described as follows.

1. Initially, all the vehicles are at the depot, and all the tasks are unserved.
2. Whenever a vehicle becomes idle (at either the beginning or upon task completion), a set of candidate tasks are selected from all the unserved tasks by a *filter* method.
3. The GP tree (essentially a priority function) is applied to calculate the priority of each candidate task, and the task with the best (smallest in this case) priority is selected to be served next.
4. If all the tasks have been served, return to depot and end the simulation. Otherwise, go back to Step 2.

In the existing GPHH approaches, the filter method identifies the candidate tasks as those expected to be feasible, i.e. their expected demand does not exceed the remaining capacity. In this paper we design a new decision making process by proposing a new filter method called the No-Early-Refill filter.

3.1 The New No-Early-Refill Filter

Algorithm 1 describes the new No-Early-Refill filter. The difference between this and the standard existing filter is that it excludes the tasks where the depot is on the expected shortest path to the task (lines 4–6) to avoid automatic, premature refilling.

Algorithm 1 The new No-Early-Refill Filter

```
1: $\Omega \leftarrow \emptyset$;
2: for each unserved task $t$ do
3:     if $\hat{d}(t) \leq \hat{Q}$ then
4:         if $\delta(\texttt{currNode}, t)$ != $\delta(\texttt{currNode}, \texttt{depot}) + \delta(\texttt{depot}, t)$ then
5:             $\Omega \leftarrow \Omega \cup t$;
6:         end if
7:     end if
8: end for
9: Return $\Omega$;
```

Figure 1 shows an example where the depot is v_0 and all the edges except (v_1, v_2) are tasks. Each edge is associated with a number denoting its deadheading cost. Suppose a vehicle has served (v_0, v_1) and is therefore located at v_1. In this state, (v_2, v_3), (v_0, v_3) and (v_0, v_4) are yet to be served. The existing filter considers all the three tasks and tends to prioritise (v_0, v_3) or (v_0, v_4) to serve next as they are closest to the vehicle's current location. The proposed No-Early-Refill filter on the other hand only considers (v_2, v_3), and can therefore serve (v_0, v_1), (v_2, v_3) and (v_3, v_0) in a single route, followed by (v_0, v_4). In doing so the new filter is able to reduce the noise introduced by these misleading tasks, making it more capable of generating solutions with smaller total cost.

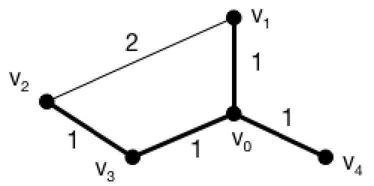


Fig. 1. An example to show the difference between filters.

3.2 The New Flood Fill Feature Terminal

It is important to efficiently handle potential route failures, reducing the extra cost caused by the failure. Intuitively, route failures occur toward the end of routes and the incurred extra cost is smaller if they occur on a task close to the depot (as a shorter recourse path is required). Therefore, given the same expected route cost, it is more desirable to serve tasks closer to the depot at the end of the route.

An example is given in Fig. 2, where v_0 is the depot, all 9 edges are tasks with an expected demand of 1, and vehicle capacity is 4. As depicted in this example, the routes R and R' have the same expected cost. However, R should be preferred over R', since the route failure in R tends to occur at the end of the route (i.e. on (v_5, v_0)), which is closer to the depot than (v_3, v_2) in R'.

Existing GPHH approaches cannot recognise this relationship. In the provided example, the GP-evolved policies tend to generate R' more often than R by preferring nearest neighbours. To address this issue, we design a new feature called *Flood Fill* (`FF`) to reflect the ability of a task to be served towards the end of the route to save the extra cost caused by the route failure.

`FF` borrows the concept of water flow dynamics, considering each edge as a pipe. When pouring water into the depot node until all the edge-pipes are uniformly full, the edges that pass a higher volume of water are easier to get to from the depot. Following this idea, we calculate the shortest path from the depot to the end of each unserved task using Djikstra's Algorithm. Then, for each task, `FF` is defined as the number of these shortest paths the task is a member of. Therefore, a smaller `FF` should be preferred.

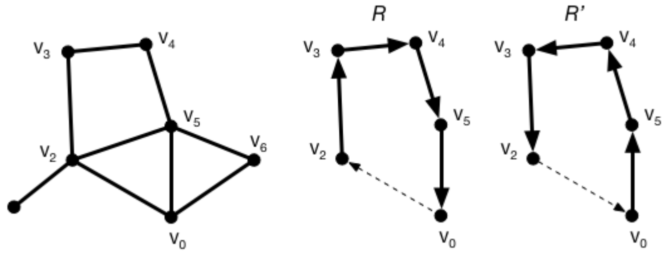


Fig. 2. An example to show the effectiveness of the new feature.

The calculation for `FF` is performed in three situations: first in the preprocessing stage, then again on the realisation of route and edge failures. Specifically, for each task t, we store a set of tasks $\Theta(t)$, which is defined as follows.

$$\Theta(t) = \{t' | \textit{either direction of } t \textit{ is on the shortest path from the depot to } t'\},$$

For all the tasks, `FF` is calculated as $\texttt{FF}(t) = |\Theta(t)|$. As task t' is served during the decision making process, each $\Theta(t)$ is updated as $\Theta(t) \setminus t'$.

4 Experimental Studies

To verify the effectiveness of the proposed No-Early-Refill filter and Flood Fill feature, we compare the following four algorithms on the *Ugdb* and *Uval* instances [16].

– GPHH [17] : the baseline GPHH algorithm with the standard filter and terminal set (given by Table 1).

– GPHH-NER: with the No-Early-Refill filter and standard terminal set.
– GPHH-FF: with the standard filter and the *extended* terminal set, which contains the standard terminals and the new `FF` feature.
– GPHH-NF: with both the No-Early-Fill filter and extended terminal set.

In the experiment, each random variable follows a truncated normal distribution, where μ is set as the value given by the static instance, and $\sigma = 0.2\mu$. For each UCARP instance, 500 sampled instances were generated independently to be the test set. A separate training set of 5 sampled instances per generation, was generated as well for the GP training process.

Table 1 gives the terminal set used in the GPHH [17]. The extended terminal set used by GPHH-FF and GPHH-NF contains `FF` as well. The function set is $\{+, -, \times, /, \max, \min\}$ (the "/" is protected, returning 1 if divided by 0). In all the compared algorithms, the population size is set to 1024, and the maximal generations is 51. The crossover/mutation/reproduction rates are set to 80%/15%/5% and the maximal depth is set to 8. All the compared algorithms were run 30 times independently, and Wicoxon rank sum test with significance level of 0.05 was conducted to test the statistical significance. This follows standard experimental norms [14, 24, 16].

Table 1. The terminals used in the GPHH.

Terminal	**Description**
`CFH`	estimated Cost From Here (the current node) to the candidate task
`CFR1`	estimated Cost From the alternative closest Route to the task
`CR`	estimated Cost to Refill (from the current node to the depot)
`CTD`	estimated Cost from the candidate task To the Depot
`CTT1`	estimated Cost from the candidate task To its closest remaining Task.
`DEM`	the estimated DEMand of the candidate task.
`DEM1`	the estimated DEMand of the closest unserved task to this candidate.
`FRT`	the Fraction of the Remaining Tasks (unserved)
`FUT`	the fraction of the Unassigned tasks
`FULL`	the FULLness of the route (current load over capacity)
`RQ`	the Remaining Capacity of the route.
`RQ1`	the Remaining Capacity for the closest alternative route.
`SC`	the Serving Cost of the candidate task.
`ERC`	a random constant number

4.1 Experimental Results

Table 2 summarises the results of the compared algorithm, and shows the mean and standard deviation of the normalised total cost of the compared algorithms on the *Ugdb* and *Uval* datasets. For normalisation, the total cost for each test instance is divided by the total cost obtained by the Path-Scanning 5 (PS5) [13] benchmark policy.

From Table 2, it can be seen that all the three newly proposed GPHH algorithms obtained better normalised total cost than GPHH. Whilst GPHH-NER

Table 2. The mean and standard deviation (in brackets) of the normalised total cost of the compared algorithms over the *Ugdb* and *Uval* datasets.

	GPHH [17]	GPHH-NER	GPHH-FF	GPHH-NF
Ugdb	0.928(0.057)	0.925(0.061)	0.927(0.059)	**0.925**(0.061)
Uval	0.828(0.048)	0.828(0.051)	0.828(0.048)	**0.825**(0.051)

performed better on the smaller *Ugdb* dataset, GPHH-FF did so on the larger *Uval* dataset with more vehicles. Finally, GPHH-NF performed the best.

For a more comprehensive statistical comparison, Table 3 shows the win-draw-lose results of the pairwise comparisons over the total 57 (23 *Ugdb* plus 34 *Uval*) instances. For example, the row-3-column-2 entry (10-44-3) shows that under the rank sum test with significance level of 0.05, GPHH-NF performed significantly better than GPHH-NER on 10 instances, and significantly worse on 3 instances. There is no statistical difference between the two algorithms on 44 instances. From the results, it is obvious that all the three newly proposed algorithms significantly outperformed the baseline GPHH on many instances. This demonstrates the effectiveness of the new No-Early-Refill filter and the `FF` feature. Note that GPHH-NER was never significantly worse than GPHH. When being used alone, the No-Early-Refill filter obtained relatively better performance than `FF`. GPHH-NER won over GPHH-FF on more instances (15 versus 6), and showed significantly better performance than GPHH on more instances than GPHH-FF (18 versus 7). Moreover, when using the new No-Early-Refill filter and `FF` simultaneously, GPHH-NF achieved much better results. For example, it significantly outperformed GPHH-NER on 10 instances, and GPHH-FF on 17 instances.

Table 3. The win-draw-lose results of the pairwise comparisons over the 57 UCARP instances.

	GPHH [17]	GPHH-NER	GPHH-FF
GPHH-NER	18-39-0	—	—
GPHH-FF	7-49-1	6-36-15	—
GPHH-NF	24-29-4	10-44-3	17-37-3

4.2 Analysis on No-Early-Refill Filter

To further analyse the effectiveness of the newly proposed No-Early-Refill Filter, we first observe how it changes the size of the candidate task set during the decision making process. Against the standard filter, the No-Early-Refill filter is stronger as it removes tasks in addition to the standard filter. Therefore, the No-Early-Refill filter tends to obtain smaller candidate tasks during the decision making process. To verify this, we observed the decision making process of GPHH, GPHH-NER and GPHH-NF on *Ugdb*23. For each of GPHH, GPHH-NER and GPHH-NF, we arbitrarily selected one run, and applied the best rule to a randomly sampled instance. For each decision making process, the candidate set size is recorded for each time the routing policy is called.

Fig. 3. Candidate task set size.

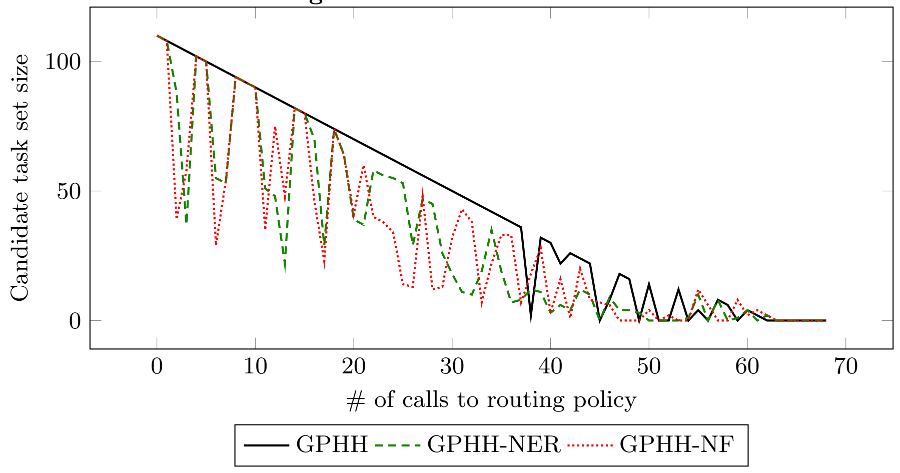


Fig. 3 shows the curves of the candidate set size of GPHH, GPHH-NER and GPHH-NF. We have examined other GP-evolved policies and other scenarios, and observed similar patterns. From the figure, it is obvious that the curve of GPHH is very smooth. This is because GPHH tends to generate early-refill routes that almost always have sufficient remaining capacity. Thus, all the unserved tasks are expected to be feasible at most decision points. The curves of GPHH-NER and GPHH-NF are below the curve of GPHH, demonstrating the effectiveness of the new filter in reducing the candidate set size. In addition, the valleys of the GPHH-NER and GPHH-NF curves show that there are usually a large fraction of candidate tasks being removed. GPHH therefore has a much higher chance of generating early-refill routes by not excluding such tasks. Finally, when averaging this across the final generation the curve of GPHH-NF is smoother than that of GPHH-NER. This demonstrates that using the No-Early-Refill filter and `FF` simultaneously can further improve the performance.

4.3 Analysis on the Flood Fill Terminal

To ensure `FF` is used to construct policies, basic high-level frequency analysis was used. This simply counted the frequency of each feature in the best policy of each generation. `FF` accounted for 7.19% of the terminals used in the *Ugdb* dataset and 6.77% on the *Uval* dataset. Whilst most other features were uniformly decreased to accompany `FF`, the rate of `CFH` use increased by 6.39% (relatively) over GPHH, suggesting the two terminals are most useful in tandem. It is worth noting that the average policy size does not significantly change between the two algorithms.

To determine whether or not `FF` was used as expected, individual analysis was performed. Presented below is an exemplary policy on the *Ugdb*2 instance.

*(+ (*max *(* CFH SC) (+ FUT (+ (- CR CTD) (+ (* FUT (* FUT SC)) (+ FUT* **FF***))))) (+ (+ CTT1 CTT1) (*max **FF** *CFH)))*

The above policy uses `FF` in two places. In both uses, a smaller `FF` will decrease the priority value, making the task more desirable. It is then not surprising that this policy has exceptional training and test performance in relation to its peers. This use trend is continued across most policies that perform well on both the training and test sets. There are policies that have an excellent training fitness yet use `FF` inverse to how we would expect. This tends to result in a disproportionately poor test fitness, showing that the GP process is not always able to appropriately utilise `FF`.

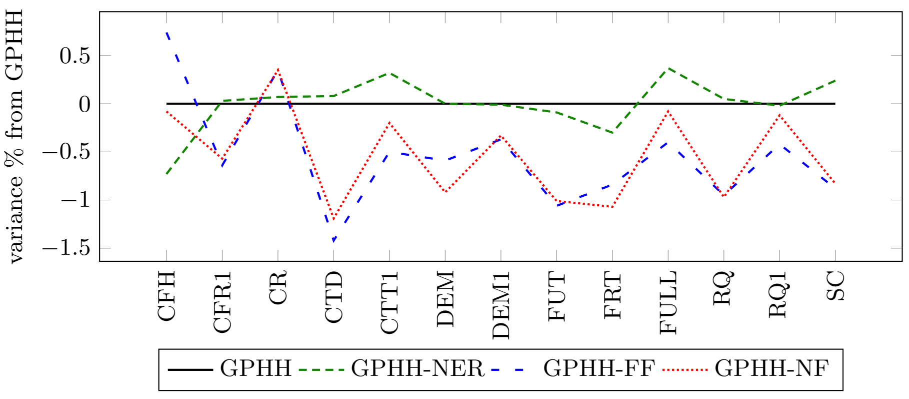
A way of improving the performance of policies using `FF` is to recalculate the flood fill map as frequently as new instance data is realised. For example, whenever the exact deadheading cost of an edge is determined. A balance between instance fidelity and computational efficiency must be struck, however.

4.4 Analysis on the integrated algorithm

Candidate set analysis was repeated to ensure the positive effects of GPHH-NER carried over. One may expect similar pool sizes in GPHH-NER and GPHH-NF, however per Fig. 3, they are significantly different. When averaging this graph across all decision processes, the GPHH-NF plot is significantly smoother than GPHH-NER. This smoothing makes logical sense when considering the effect `FF` has on the simulation, where smoother traversal of the graph is encouraged.

Terminal frequency analysis was also performed to ensure `FF` was used in a similar manner to that of GPHH-FF. This was mostly true, with GPHH-NF utilising the feature slightly more; 7.01%. Fig. 4 shows the terminal use variance from GPHH on the standard terminal set (i.e. algorithm % value minus GPHH % value). A notable difference is the usage of `CFH`, showing the new filter in GPHH-NER decreases the demand for the feature - a concept reinforced by comparison of GPHH-NER and GPHH.

Fig. 4. Standard terminal use variation on GPHH



5 Conclusions and Future Work

Our motivations with this paper have been to ensure vehicle capacity is wholly utilised before replenishment and to encourage the prioritisation of hard to access tasks. This paper has presented two new techniques to meet these motivations and improve the performance of the GPHH method. Firstly, a filtering method that removes tasks from the possible selection pool if the fastest route to said task passes the depot was shown to significantly outperform the standard GPHH benchmark on 18 of the 57 tested instances. Secondly, the introduction of a new flood fill value to the possible terminal set improved on 7 of the tested instances. Additionally, when used together, these improvements further increased the performance over the benchmark of [14] on 24 of the tested instances. Note that improvement of this merged algorithm was not limited or restricted to the instances on which the two sub-algorithms performed well - together, the two parts became a distinct whole.

Analysis was done on the terminal use and pool-size in relation to the GPHH benchmark, highlighting some important anomalies between instance performance. This highlighted the need for further research into the effects algorithms have on particular instance characteristics. From this research, it is clear that very particular algorithms react, often erratically, to the specific nature of the instance in question. Discovering what exactly these critical topological features are, and deciding which algorithmic features to use in each given environment is certainly an interesting area for future research we believe worth exploring.

References

1. Amponsah, S., Salhi, S.: The investigation of a class of capacitated arc routing problems: The collection of garbage in developing countries. Waste Management **24**(7), 711–721 (2004)
2. Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M.: Automated design of production scheduling heuristics: A review. IEEE Transactions on Evolutionary Computation **20**(1), 110–124 (2016)
3. Burke, E.K., Hyde, M., Kendall, G., Woodward, J.: A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. IEEE Transactions on Evolutionary Computation **14**(6), 942–958 (2010)
4. Christiansen, C., Lysgaard, J., Wøhlk, S.: A branch-and-price algorithm for the capacitated arc routing problem with stochastic demands. Operations Research Letters **37**(6), 392–398 (2009)
5. Defryn, C., Srensen, K., Cornelissens, T.: The selective vehicle routing problem in a collaborative environment. European Journal of Operational Research **250**(2), 400 – 411 (2015)
6. Eglese, R.W., Li, L.Y.O.: A tabu search based heuristic for arc routing with a capacity constraint and time deadline. Meta-Heuristics: Theory and Applications pp. 633–649 (1996)
7. Fleury, G., Lacomme, P., Prins, C., Ramdane-Chérif, W.: Improving robustness of solutions to arc routing problems. Journal of the Operational Research Society **56**(5), 526–538 (2005)
8. Golden, B., Dearmon, J., Baker, E.: Computational experiments with algorithms for a class of routing problems. Computers & Operations Research **10**, 47 – 59 (1983)
9. Golden, B., Wong, R.: Capacitated arc routing problems. Networks **11**(3), 305–315 (1981)

10. Handa, H., Chapman, L., Yao, X.: Dynamic salting route optimisation using evolutionary computation. In: IEEE Congress on Evolutionary Computation. pp. 158–165 (2005)
11. Handa, H., Chapman, L., Yao, X.: Robust route optimization for gritting/salting trucks: a cercia experience. IEEE Computational Intelligence Magazine **1**(1), 6–9 (2006)
12. Hertz, A., Laporte, G., Mittaz, M.: A Tabu Search Heuristic for the Capacitated Arc Routing Problem, vol. 48, pp. 129 – 135. Operations Research, Linthicum, MA (2000)
13. Lacomme, P., Prins, C., Ramdane-Cherif, W.: Competitive memetic algorithms for arc routing problems. Annals of Operations Research **131**(1), 159–185 (2004)
14. Liu, Y., Mei, Y., Zhang, M., Zhang, Z.: Automated heuristic design using genetic programming hyper-heuristic for uncertain capacitated arc routing problem. In: Proceedings of GECCO. pp. 290–297. ACM (2017)
15. Mei, Y., Tang, K., Yao, X.: Improved memetic algorithm for capacitated arc routing problem. IEEE Congress on Evolutionary Computation pp. 1699 – 1706 (2009)
16. Mei, Y., Tang, K., Yao, X.: Capacitated arc routing problem in uncertain environments. In: IEEE Congress on Evolutionary Computation. pp. 1–8 (2010)
17. Mei, Y., Zhang, M.: Genetic programming hyper-heuristic for multi-vehicle uncertain capacitated arc routing problem. ACM Genetic and Evolutionary Computation Conference (GECCO) (2017)
18. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex & Intelligent Systems **3**(1), 41–66 (2017)
19. Speranza, M., Fernandez, E., Roca-Riu, M.: The shared customer collaboration vehicle routing problem. European Journal of Operational Research **265**(3), 1078 – 1093 (2016)
20. Tsutsui, S., Wilson, G.: Solving capacitated vehicle routing problems using edge histogram based sampling algorithms. In: Proceedings of the 2004 Congress on Evolutionary Computation. vol. 1, pp. 1150–1157 (2004)
21. Ulusoy, G.: The fleet size and mix problem for capacitated arc routing. European Journal of Operational Research **22**(3), 329 – 337 (1985)
22. Wang, J., Tang, K., Lozano, J.A., Yao, X.: Estimation of the distribution algorithm with a stochastic local search for uncertain capacitated arc routing problems. IEEE Transactions on Evolutionary Computation **20**(1), 96–109 (2016)
23. Wang, J., Tang, K., Yao, X.: A memetic algorithm for uncertain capacitated arc routing problems. In: 2013 IEEE Workshop on Memetic Computing. pp. 72–79 (2013)
24. Weise, T., Devert, A., Tang, K.: A developmental solution to (dynamic) capacitated arc routing problems using genetic programming. In: Proceedings of GECCO. pp. 831–838. ACM (2012)