Hyper-heuristic Operator Selection and Acceptance Criteria

Richard J. Marshall[1](✉), Mark Johnston[1], and Mengjie Zhang[2]

[1] School of Mathematics, Statistics and Operations Research,
Victoria University of Wellington, Wellington, New Zealand
richardj.marshall@xtra.co.nz, mark.johnston@msor.vuw.ac.nz
[2] School of Engineering and Computer Science,
Victoria University of Wellington, Wellington, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

Abstract. Earlier research has shown that an adaptive hyper-heuristic can be a successful approach to solving combinatorial optimisation problems. By using a pairing of an operator (low-level heuristic) selection vector and a solution acceptance criterion, an adaptive hyper-heuristic can manage development of a "good" solution within an unseen low-level problem domain in a commercially realistic computational time. However not all selection vectors and solution acceptance criteria pairings deliver competitive results when faced with differing problem instance features and computational time limits. We evaluate pairings of six different operator selection vectors and eight solution acceptance criteria, and monitor the performance of the adaptive hyper-heuristic when applying each pairing to a set of Capacitated Vehicle Routing Problem instances of the same size but with different features. The results show that a few pairings of operator selection vector and acceptance criterion perform consistently well, while others require a longer computational time to deliver competitive results. We also investigate some of the features of a problem instance that may influence the performance of the selection vector and acceptance criterion pairings.

1 Introduction

Traditional methods of solving combinatorial optimisation problems use algorithms and heuristics, such as a branch-and-bound algorithm [4] or meta-heuristic search (e.g. tabu search [5]). These methods can achieve good results but often require detailed domain information and can be complex and time consuming to design and execute. A hyper-heuristic is useful where a more general (domain independent) approach is required. The term hyper-heuristic was defined by Cowling et al. [2] as "heuristics to choose heuristics". In this respect, we use a hyper-heuristic to select and execute operators (heuristics) from an unseen set of low-level (domain specific) operators, which in turn incrementally build and/or modify a solution to each problem instance. Understanding what makes a particular hyper-heuristic efficient and effective would enable the trade-off between computational speed and quality of the result to be *managed* when faced with larger problem instances and more complex problem domains.

© Springer International Publishing Switzerland 2015
G. Ochoa and F. Chicano (Eds.): EvoCOP 2015, LNCS 9026, pp. 99–113, 2015.
DOI: 10.1007/978-3-319-16468-7_9

The winning entry of the First Cross-domain Heuristic Search Challenge (CHeSC) [8] in 2011 was an adaptive hyper-heuristic designed by Misir et al. [7]. Marshall et al. [6] illustrate that even a simplified version of the adaptive hyper-heuristic designed by Misir et al. [7] performs well when applied to unseen problems in seven different combinatorial optimisation problem domains. The key components to the adaptive hyper-heuristic are the *operator selection vector*, which determines which operator to apply next, and the *solution acceptance criterion* which determines whether a new solution is retained or discarded. In [6], the simplified adaptive hyper-heuristic used a single operator selection vector and a solution was only accepted if it was better than or equally as good as the solution it may replace. This paper investigates whether providing a wider choice of operator selection vector and solution acceptance criteria can improve the effectiveness of the adaptive hyper-heuristic. We increase the number of possible operator selection vectors to six, and the number of solution acceptance criteria to eight.

The remainder of this paper is organised as follows. A brief background is given in Sect. 2, including an overview of the adaptive hyper-heuristic. Section 3 describes the operator selection vectors and solution acceptance criteria. The experimental design, results and discussion are in Sects. 4 and 5. Finally, Sect. 6 gives our conclusions.

2 Background

This paper uses an adaptive hyper-heuristic (AdaptiveHH) compatible with the HyFlex (**Hy**per-heuristic **Flex**ible) framework [9]. We focus on using a Capacitated Vehicle Routing Problem (CVRP) [12] domain. However, since the hyper-heuristic has no problem domain dependent processes, the hyper-heuristic can readily be applied to other problem domains. Importantly, the hyper-heuristic has no knowledge of the size or features of the problem instance it is working on. This means that the specified computational time limit may be excessive or insufficient to arrive at a "good" solution to the problem instance. The hyper-heuristic must be able to make the best use of the available time and, ideally, terminate processing early when the available time is excessive.

2.1 Adaptive Hyper-heuristic

The AdaptiveHH in this paper is a simplified version of the hyper-heuristic developed by Misir et al. [7]. Conceptually, AdaptiveHH iteratively selects and applies an unseen operator from the problem domain. The resulting solution is then accepted or discarded based on the acceptance criterion specified by AdaptiveHH. AdaptiveHH requires a number of parameters which set the computational time limit, the number of intermediate decision points (phases), and the choice of operator selection vector and acceptance criterion to use (see Fig. 1). Parameters also set the rules about how AdaptiveHH responds if progress towards improving the current solution is stalled.

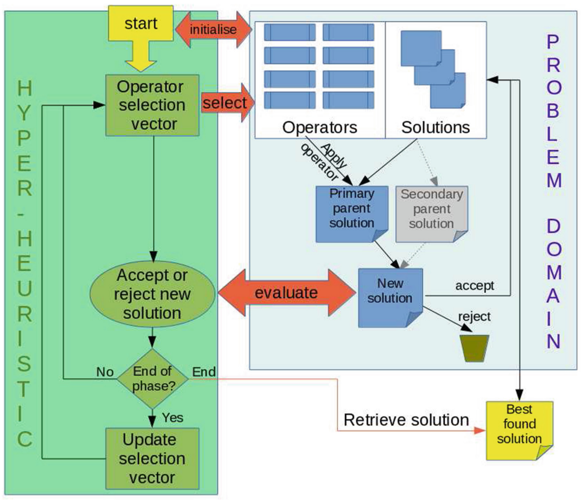

Fig. 1. Overview of how an adaptive hyper-heuristic interacts with a low-level problem domain across the domain barrier.

The two main components of AdaptiveHH are:

1. **The Operator Selection Vector.** This vector is used to select the next operator to apply. The vector is updated at the start of each phase based on the performance of the operator in the preceding phase(s). It gives the probability of each operator being selected and applied to the current solution.
2. **The Solution Acceptance Criteria.** Once an operator modifies a solution to create a new solution, the hyper-heuristic needs to decide whether to accept (retain) or discard the new solution.

2.2 HyFlex Framework

The HyFlex framework [9] was originally developed in 2011 for the First Cross-domain Heuristic Search Challenge (CHeSC) [8]. The framework includes six in-built combinatorial optimisation problem domains. Associated with each in-built problem domain is a set of between 8 and 15 unseen low-level operators (heuristics). Each set contains at least one operator belonging to each of the four defined operator types: *mutation*, *ruin-recreate*, *local search* and *crossover*. A crossover operator swaps parts of one solution with another solution in an attempt to create a better solution.

Each operator can use (if appropriate) the two HyFlex parameters α and β, where $(0 \leq \alpha, \beta \leq 1)$. The Intensity of Mutation parameter, α, affects the scale of any mutation or ruin operation, e.g., 0.5 would mean half the current solution would be altered by an operator using this parameter. The Depth of Search parameter, β, defines a range or number of repetitions an operator will undertake to find an improved solution in a single execution of the operator.

Each operator is only visible to the hyper-heuristic to the extent allowed by the HyFlex [9] specifications. Operator visibility is restricted to the following properties:

1. **Operator Type.** A mandatory attribute of each operator contained within a HyFlex problem domain. There are four defined operator types:
   (a) **Mutation** operators add or reposition an element in a solution. Operators of this type would generally only involve simple manipulations requiring a short computational time which is only marginally affected by the size of the problem instance.
   (b) **Ruin-Recreate** operators destroy a segment of an existing solution, chosen by the operator implementation, and then rebuild the segment to form a new solution. These operators are more complex than a mutation operator and typically require a longer computational time. The computational time may vary substantially depending on the size of the problem instance.
   (c) **Local Search** operators define and search a solution neighbourhood for improvements. These operators generally apply a degree of logic to the search so can be expected to have a higher chance of improving a solution, or verify no further local improvements are possible, than the other operator types. However, the computational time may be much longer, and could escalate polynomially (or worse) as the problem instance size increases.
   (d) **Crossover** operators combine elements of two current solutions to form a new solution. The computational time of a crossover operator varies but is often similar to a ruin-recreate operator.
2. **Uses Intensity of Mutation.** An indicator to show whether this operator uses the global Intensity of Mutation, α, parameter.
3. **Uses Depth of Search.** An indicator to show whether this operator uses the global Depth of Search, β, parameter.
4. **Call Record.** The number of times the operator has been executed during a run is calculated and is visible to the hyper-heuristic on demand.
5. **Call Time Record.** The aggregate of the execution time of each operator during a run is recorded and is visible to the hyper-heuristic on demand.

3 The Method

We test the effectiveness of AdaptiveHH by rating each solution generated against the best solution objective value achieved within the computational time limit. We use different pairings of operator selection vector and acceptance criterion. There are 48 possible pairings of operator selection vector (6) and solution acceptance criteria (8).

3.1 Operator Selection Vector Design

AdaptiveHH operates for a specified time limit which is broken down into phases. The operator selection vector is updated at the end of each phase. The choice of selection vector and acceptance criterion is fixed at the beginning of the run and is not altered during the run. The selection vector consists of an array of operators, each with a probability of selection. In the initial selection vector (regardless of type) all operators have an equal probability of selection.

We follow the example of Misir et al. [7] and allow some of the selection vectors described below to exclude operators for one or more phases (i.e. the selection probability is zero). The number of phases an operator is excluded is based on a performance penalty. The first time an operator is excluded the performance penalty is set to one. This means the operator is readmitted to the selection vector at the end of the next phase (i.e. one phase exclusion) with a probability of 0.01 prior to normalisation. If the operator is immediately excluded again during the vector update process at the end of the readmission phase, the performance penalty, and hence the number of exclusion phases, is increased by one. Should the operator be readmitted and survive the vector update process into the succeeding phase, then the performance penalty is reset to one.

The AdaptiveHH reported in [6] used only the Basic Selector. The operator selection vectors are of our own design, but use components of the single selection vector used by Misir et al. [7].

1. **[FS] Fixed Selector:** The initial vector is not altered during the run, so provides a benchmark against which other selection vectors can be measured. All operators have an equal probability of selection regardless of performance.
2. **[BS] Basic Selector:** Updates probabilities by evaluating the success rate of each operator, r_i, since the start of the run:

$$r_i = \frac{\text{number of improvements}_i}{\text{number of calls}_i}$$

   This vector does not exclude operators and sets a minimum probability of selection as 0.001 prior to normalisation.
3. **[P1] Phase Selector (1):** Updates probabilities by evaluating the success rate of the each operator, r_i (as per [BS]), in the most recent phase. During the update process a threshold is set equal to $\frac{1}{3}$ of the success rate of the best performing operator, r_{best}, in that phase. If $r_i \geq \frac{r_{best}}{3}$ it is included in the selection vector for the next phase with a probability of r_i, minimum 0.01, prior to normalisation. Operators where $r_i < \frac{r_{best}}{3}$ are excluded from the vector for the number of phases determined by their performance penalty.
4. **[P2] Phase Selector (2):** Updates probabilities by evaluating the success rate of each operator, r_i (as per [BS]), in the most recent phase. This vector does not exclude operators and sets a minimum probability of selection at 0.001 prior to normalisation. It differs from the Basic Selector in that this selection vector only considers performance during the most recent phase.

5. **[T1] Time Weighted Phase Selector (1):** The time weighted selector uses a time weight, w_i, to penalise slower operators. This is calculated using the average operator execution time, $averageOpTime$, during the preceding phase:
$$w_i = \sqrt{\frac{averageOpTime_i}{averageOpTime_{fastest}}}$$
The time weighted success rate of each operator, r_i, in the most recent phase is evaluated:
$$r_i = \frac{\text{number of improvements}_i}{w_i \times \text{number of calls}_i}$$
During the update process a threshold is set equal to $\frac{1}{3}$ of r_{best}. If $r_i \geq \frac{r_{best}}{3}$ it is included in the selection vector for the next phase with a probability of r_i, minimum 0.01, prior to normalisation. Operators where $r_i < \frac{r_{best}}{3}$ are excluded from the vector for the number of phases determined by their performance penalty.
6. **[T2] Time Weighted Phase Selector (2):** Calculation of the time weight, w_i, and success rates, r_i, are identical to that described for [T1]. For this selector all r_i are ranked highest to lowest, including those excluded from the selection vector ($r_i = 0$). A threshold, T, is set equal to the r_i of the operator ranked $\frac{NumberOfOperators}{2}$ (T may be zero). If $r_i \geq T$, it is included in the selection vector for the next phase with a probability weighting of $\frac{1}{\text{rank}}$, prior to normalisation.

3.2 Acceptance Criteria Design

Each application of an operator takes a current solution and modifies it to create a new solution. The new solution is then considered for acceptance into the small population of solutions. If the new solution is not accepted then it is discarded. If the new solution is at least as good as the solution it will replace, then it is automatically accepted into the population of solutions regardless of the acceptance criteria specified by the hyper-heuristic. The following eight acceptance criteria are those proposed by Sabar et al. [10] with minor modifications. As far as possible we have retained the labels and arbitrary parameter values proposed by Sabar et al. [10] and only made changes which are necessary to satisfy the HyFlex framework [9] constraints. In all cases, the new solution is compared to the solution it will replace (if accepted) in the population of solutions.

1. **[IO] Improving or Equal Only:** Only improving (better objective value) or equally good solutions are accepted. All other solutions are discarded.
2. **[AM] Accept Move:** All new solutions are accepted.
3. **[SA] Simulated Annealing:** Non-improving solutions are accepted with a probability $e^{-\delta/t}$, where δ is the change in the objective value between the old and new solutions. The "temperature", t, is $0.5 \times S_{best} \times 0.85^{phase-1}$, where S_{best} is the current best solution objective value [1,11]. The probability of a non-improving solution being accepted decreases as (a) the change in objective value increases and (b) as time progresses.

4. **[MC] Exponential Monte Carlo:** Non-improving solutions are accepted with a probability $e^{-\delta}$, where δ is the change in the objective value between the old and new solutions. The probability of a non-improving solution being accepted decreases as the change in objective value, δ, increases.
5. **[RR] Record to Record Travel:** Non-improving solutions are accepted if the new solution has an objective value less than or equal to $1.03 \times S_{best}$, where S_{best} is the current best solution objective value [3].
6. **[GD] Great Deluge:** Non-improving solutions are accepted if the new solution has an objective value less than or equal to $(1 + 0.85^{phase-1}) \times S_{best}$, where S_{best} is the current best solution objective value [3]. The probability of a non-improving solution being accepted decreases as time progresses.
7. **[NA] Naïve Acceptance:** Non-improving solutions are accepted with 0.5 probability.
8. **[AA] Adaptive Acceptance:** Non-improving solutions are accepted with a probability $1 - \frac{1}{C}$, where $C > 0$ is a counter which increments every 10,000 consecutive operator calls without an improvement in the objective value of the best solution found so far. The counter is reset to 1 each time an improved best solution objective value is found. The probability of a non-improving solution being accepted increases when the search for better solutions reaches a plateau and new best found solutions become harder to find.

4 Experimental Design

The experiments use our own implementation of a CVRP domain [12] compatible with the HyFlex [9] framework. We create 50 random 80-node (79 customers + 1 depot) problem instances requiring a minimum of between 3 and 19 routes. Each problem instance is randomly created using an 80×80 grid. Each instance contains three nodes at fixed locations (see Fig. 2), one of which is the depot, and the other 77 nodes at randomly generated locations. Vehicle capacity is fixed at 1,000 units and each customer's demand is a randomly generated integer with an upper bound ranging from 5 % to 45 % (randomly set for each instance) of the vehicle capacity, with a minimum demand of 1 unit.

We modify the twelve low-level operators proposed by Walker et al. [13] for a CVRP-with-time-windows domain by removing the time window elements from each operator. There are 4 mutation, 2 ruin-recreate, 4 local search and 2 crossover operator types (see Sect. 2.2). The size of the population of solutions is set at six. The hyper-heuristic is only provided with the number of operators of each operator type and has no knowledge of the actual function each operator performs.

We seek to determine:

1. Whether there are particular pairings of operator selection vector and acceptance criterion which consistently perform well or poorly compared to other pairings in arriving at a "good" solution within a short computational time. We examine how each pairing affects the frequency with which each operator type is selected.

2. Whether the location of the depot in relation to the customers influences the consistency and quality of the solutions. To this end we take a problem instance (see Fig. 2) and relocate the depot by swapping the grid coordinates of the depot with one of two customers highlighted. The problem instance is otherwise unchanged. The alternative depot locations are chosen so that the depot is geographically: (a) central, (b) off-centre, and (c) remote.
3. Although we use CVRP instances of the same size, the differing customer demand values mean solutions require a minimum number of routes ranging from 3 to 19. We examine the influence the number of routes has on the performance of the operator selection vector and acceptance criterion pairings. This, and the preceding objective, will determine whether the structure of the problem instance affects performance of the pairings.

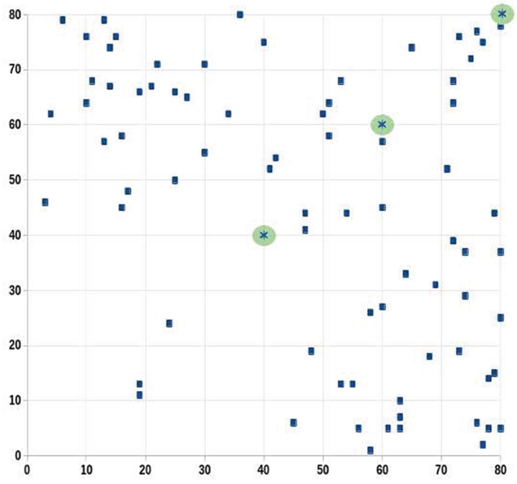

Fig. 2. Randomly generated 80-node problem instance on an 80×80 grid, showing the 3 alternative depot locations (highlighted).

We compare the quality of the results from 30 replications on a set of 50 randomly generated 80 node CVRP instances. We rate individual solutions against the best solution found during the batch of runs (typically 1,440 runs, being 30 replications of 48 pairings) using the following formula (lower ratings are better).

$$\text{rating}_i = \left(\frac{100 \times (\text{solution}_i - \text{solution}_{best})}{\text{solution}_{best}} \right)^2$$

This provides an indication of the relative performance of each pairing compared to its peers. We use the square of the result to increase the apparent difference between results and increase the penalties for poor solutions.

We conduct 25-phase (see Sect. 2.1) experiments using three different depot locations on 50 CVRP instances. We measure pairing performance using computational time limits of 1, 5, 15, 30, 60, 120 and 300 s. The hyper-heuristic we use contains a reinitialisation mechanism if no improving solutions are found for 10,000 consecutive operator calls. It also contains an early termination condition should there still be no improving solutions for two consecutive phases. The purpose of this mechanism is to allow processing to be halted when the hyper-heuristic detects there is a very low likelihood of making further improvements to the best found solution so far.

5 Results and Discussion

Table 1 and Fig. 3 show the results for all pairings from the batches using a 60 s computational time limit for each depot location. Due to space constraints, Tables 2 and 3 only show results from the five best and five worst performing pairings identified in Table 1. Widely differing customer demand values mean the 50 CVRP instances require a minimum of between 3 and 19 routes to service all customers. Table 2 compares the performance of pairings on problem instances where the minimum number of routes is small (3–5 routes), medium (6–13 routes) and large (14–19 routes). Table 3 shows the change in performance over different computational time limits. In Table 3 the performance is measured against the best solution found in any batch for each CVRP instance and depot location. Early terminations only affect the data when allowing a 300 s computational time limit. A negligible number (<0.1 %) of early terminations occurred with a 120 s time limit, and none with the shorter time limits.

Table 1 illustrates the difference in performance when the depot is at different locations. While all pairings provide better results when the depot is located centrally compared to off-centre, the better performing pairs generally

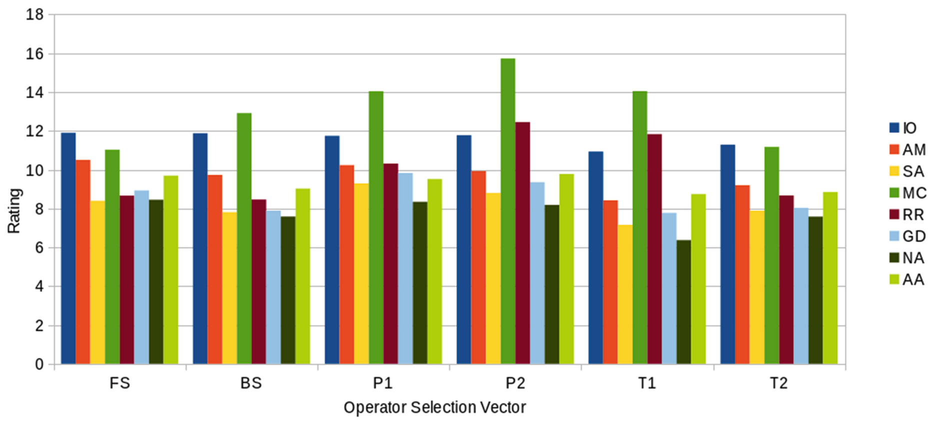

Fig. 3. Comparison of acceptance criteria and selection vector performance ratings (see Sect. 4) during 60 s runs shown in Table 1. Lower ratings are better.

Table 1. Average rating (see Sect. 4) of each selection vector and acceptance criteria pairing over 3 depot locations × 30 replications × 60 s runs on 50 randomly generated CVRP instances (80 nodes, 3 depot locations (see Fig. 2)). Lower ratings are better. Best five performing pairs in bold; five worst in italics.

Acceptance Criteria	Selection Vector	Central Depot	Off-centre Depot	Remote Depot	Average Rating	Std.Dev.
IO	FS	*10.33*	12.35	13.02	11.90	13.62
IO	BS	10.09	12.50	13.04	11.88	13.53
IO	P1	9.96	12.29	12.99	11.75	13.50
IO	P2	10.09	12.29	12.96	11.78	13.38
IO	T1	9.60	11.21	11.99	10.93	12.50
IO	T2	9.74	11.63	12.50	11.29	12.85
AM	FS	9.01	11.89	10.61	10.51	12.42
AM	BS	8.09	11.09	10.02	9.73	11.37
AM	P1	8.19	11.74	10.79	10.24	12.27
AM	P2	7.51	11.57	10.72	9.93	11.97
AM	T1	6.25	10.17	8.86	8.43	10.84
AM	T2	7.35	10.78	9.48	9.20	10.95
SA	FS	6.81	9.25	9.13	8.40	10.31
SA	BS	**5.93**	**8.56**	8.94	7.81	10.36
SA	P1	7.19	10.34	10.36	9.30	11.52
SA	P2	6.74	9.65	10.02	8.80	10.74
SA	T1	**5.59**	**8.06**	**7.83**	**7.16**	9.60
SA	T2	6.10	8.80	8.77	7.89	10.35
MC	FS	9.75	10.96	12.38	11.03	12.54
MC	BS	*11.02*	13.03	*14.70*	*12.92*	14.28
MC	P1	*11.92*	*14.34*	*15.87*	*14.04*	15.28
MC	P2	*13.49*	*16.35*	*17.36*	*15.73*	17.04
MC	T1	*11.77*	*15.01*	*15.37*	*14.05*	15.60
MC	T2	9.99	11.05	12.49	11.18	12.96
RR	FS	6.77	9.46	9.76	8.66	10.88
RR	BS	6.50	9.25	9.63	8.46	10.76
RR	P1	7.73	11.30	11.91	10.31	12.27
RR	P2	9.21	*13.85*	*14.29*	*12.45*	14.21
RR	T1	9.29	*13.05*	13.16	11.83	13.55
RR	T2	6.55	9.30	10.17	8.68	11.01
GD	FS	7.66	9.33	9.81	8.93	11.14
GD	BS	6.47	**7.98**	9.22	7.89	10.29
GD	P1	8.15	10.35	11.00	9.83	11.92
GD	P2	7.57	9.82	10.69	9.36	11.31
GD	T1	6.15	8.74	**8.42**	**7.77**	10.32
GD	T2	6.72	**8.49**	8.88	8.03	10.33
NA	FS	6.71	9.66	8.99	8.45	10.61
NA	BS	**5.96**	8.68	**8.11**	**7.59**	10.01
NA	P1	6.38	9.63	9.04	8.35	10.51
NA	P2	6.33	9.35	8.86	8.18	10.20
NA	T1	**4.58**	**7.60**	**6.95**	**6.37**	8.88
NA	T2	**5.88**	8.64	**8.22**	**7.58**	9.84
AA	FS	8.58	10.60	9.92	9.70	11.58
AA	BS	8.08	9.48	9.51	9.03	10.85
AA	P1	8.28	10.31	9.96	9.52	11.55
AA	P2	8.12	10.74	10.49	9.78	11.91
AA	T1	7.04	9.93	9.26	8.75	11.29
AA	T2	7.62	9.55	9.39	8.85	11.04

show improving performance when the depot is moved even further away from the centre. In contrast, the poorer performing pairings generally show neutral to worsening results the farther the depot is located from the geographic centre. This highlights that the size of the problem instance is not the only factor influencing the performance of the hyper-heuristic. Table 1 also confirms that there is an inter-dependency between the operator selection vector and the acceptance criterion and it is insufficient to separately evaluate each, even though they carry out different functions.

As shown in Table 1, some pairings, such as [SA][T1] and [NA][T1], consistently perform better than other pairings. The pairings using the [MC] and [IO] acceptance criteria generally perform poorly and require a longer computational time to achieve results competitive with the better performing pairings.

A possible cause of this difference is the diversity in the population of solutions. Pairings using the [IO] acceptance criterion, and to a lesser extent [MC], work with a smaller diversity of interim solutions compared to other forms of acceptance criteria. This means that time and effort are not lost on improving low quality solutions that may never become the best solution in the current population of interim solutions. This is a useful trait if the computational time limit is very short, since effort is directed towards improving a better quality solution. On the other hand, accepting only improving or equally good solutions can cause the population of solutions to stagnate and eventually become clones of the best found solution. Once this stage is reached the crossover operators become ineffective and there is a tendency for the process to stall. The hyper-heuristic has a mechanism to reinitialise the population of solutions in the event of stalling, but this is only effective if the selection vector and acceptance criterion pairing can avoid regenerating the same set of solutions.

An increase in the number of routes (see Table 2) as well as the relative second location of the depot are influencing factors as well. However, Tables 1 and 2 also

Table 2. Comparison of the ratings (see Sect. 4) of the five best and five worst performing pairings from Table 1 on CVRP instances requiring a small (15 instances), medium (18 instances) or large (17 instances) minimum number of routes.

Acceptance	Selection	3–5 routes	6–13 routes	14–19 routes
NA	T1	5.84	7.43	5.54
SA	T1	6.30	8.25	6.53
NA	T2	5.30	8.77	7.93
NA	BS	5.30	8.71	8.01
GD	T1	6.41	9.01	7.35
RR	P2	7.94	12.97	15.29
MC	BS	8.24	14.16	15.03
MC	P1	9.56	15.27	16.03
MC	T1	11.84	15.99	12.45
MC	P2	11.55	16.90	17.56

Table 3. Average rating of five best and five worst performing pairings from Table 1 on 50 CVRP instances × 3 depot locations × 30 replications, when allowing a different computational time limit (in seconds). All ratings are measured against the best solution to each instance (and depot location) found during any of the seven batches.

Acceptance	Selection	1	5	15	30	60	120	300
NA	T1	45.27	18.85	12.39	9.25	6.88	5.37	4.07
SA	T1	48.08	20.47	14.14	10.52	7.72	5.88	4.28
NA	T2	888.03	22.97	13.96	10.81	8.19	6.34	4.67
NA	BS	57.72	19.90	13.79	10.89	8.19	6.16	4.49
GD	T1	47.08	22.33	14.97	11.27	10.55	6.40	4.63
RR	P2	46.62	25.11	18.92	15.88	13.29	10.72	8.16
MC	BS	60.19	23.67	18.64	16.06	13.73	11.77	9.69
MC	P1	52.42	29.00	21.62	18.00	14.85	12.35	9.93
MC	T1	54.62	28.95	21.95	18.82	14.88	12.03	9.83
MC	P2	51.74	30.51	23.19	20.01	16.62	13.53	10.96

show that the **relative** performance of operator selection vector and acceptance criterion pairings compared to other pairings is not greatly altered by the number of routes or depot location. A better performing pairing will consistently deliver higher quality solutions than poorer performing pairings regardless of the depot location or the minimum number of routes.

Table 3 shows the performance of each pairing improves with a longer computational time limit, but not all improve at the same rate. The [NA][T2] pairing performs poorly with the 1 s computational time limit but well with longer time limits, indicating a minimum time limit per phase is necessary for some pairings before the operator selection vector update process can be effective. In these experiments the improvement in the performance of the better performing pairings appears to be reaching a plateau with a 300 s time limit. However, the poorer performing pairings show a non-trivial improvement in performance between 120 and 300 s time limits, suggesting a longer computational time may produce further improvements.

Table 4 illustrates how different pairings of operator selection vector and acceptance criterion affect the frequency with which particular operator types are called. The number of calls illustrates how the more aggressive of the two time weighted selection vectors, [T1], biases operator selection towards the faster mutation and crossover operators and away from the slower local search operators. The second time weighted selection vector, [T2], maintains a more balanced selection approach. The Fixed Selector [FS] reflects the 4:2:4:2 balance between the four operator types in the CVRP domain.

The time-weighted selectors favour the faster mutation operators at the expense of the slower search operators. This is a trade-off between speed and quality. Table 4 shows that a large number of operator calls is not critical to the

Table 4. Average number of operator calls, success rate (r_i, as defined in Sect. 3.1) and mix of operator type selection between the six selection vectors (SV) and the Naïve Acceptance [NA], Exponential Monte Carlo [MC] and Improving or Equal Only [IO] acceptance criteria (AC) during experiments using a 60 seconds computational time limit. Best performing pairs (from Table 1) in bold, worst in italics.

AC	SV	Num. calls	Success	Mutation	Ruin-rec.	Search	Crossover
NA	FS	236,900	11.82 %	33.34 %	16.66 %	33.33 %	16.67 %
NA	**BS**	**168,900**	**8.12 %**	**12.65 %**	**15.44 %**	**47.01 %**	**24.89 %**
NA	P1	138,300	5.53 %	5.86 %	10.41 %	54.29 %	29.44 %
NA	P2	231,000	4.71 %	4.40 %	3.57 %	29.40 %	62.63 %
NA	**T1**	**749,400**	**5.78 %**	**2.34 %**	**2.88 %**	**3.53 %**	**91.24 %**
NA	**T2**	**187,500**	**6.82 %**	**17.08 %**	**14.81 %**	**45.22 %**	**22.89 %**
MC	FS	220,800	1.43 %	33.36 %	16.64 %	33.34 %	16.66 %
MC	*BS*	*253,600*	*2.38 %*	*7.11 %*	*5.71 %*	*30.27 %*	*56.91 %*
MC	*P1*	*246,100*	*2.54 %*	*3.41 %*	*2.61 %*	*30.81 %*	*63.17 %*
MC	*P2*	*269,600*	*2.70 %*	*3.30 %*	*1.64 %*	*23.59 %*	*71.47 %*
MC	*T1*	*821,400*	*2.73 %*	*1.68 %*	*0.84 %*	*1.68 %*	*95.80 %*
MC	T2	323,800	1.79 %	51.81 %	11.63 %	22.07 %	14.49 %
IO	FS	229,900	0.16 %	33.32 %	16.66 %	33.35 %	16.67 %
IO	BS	196,100	0.14 %	18.31 %	18.28 %	39.00 %	24.41 %
IO	P1	201,000	0.15 %	25.29 %	18.55 %	35.22 %	20.94 %
IO	P2	191,800	0.16 %	22.32 %	18.20 %	33.23 %	26.25 %
IO	T1	512,100	0.18 %	34.95 %	8.69 %	8.55 %	47.81 %
IO	T2	373,200	0.16 %	56.22 %	12.51 %	18.85 %	12.41 %

quality of the solution. This table also illustrates the lower number of calls made to crossover type operators when the [IO] acceptance criteria is used, reflecting the reduced effectiveness of these operators in this situation. In contrast, the time weighted selector [T1] almost exclusively uses the crossover operator with both the Naïve Acceptance [NA] and Exponential Monte Carlo [MC] acceptance criteria. With [NA], the resulting solutions are among the best, while with [MC] they are among the worst. This can be explained by the difference in the diversity of the population of solutions, as reflected in the relative operator call success rates. However other factors such as parameter values and the number of early terminations (42 % during 300 s time limit) due to best found solutions no longer improving, may also influence the difference in overall solution quality.

6 Conclusions

When comparing the results in Tables 1, 2 and 3 we deduce the following about the operator selection vector and acceptance criteria pairings.

1. Generally perform well:
   (a) Time Weighted Phase Selectors [T1] and [T2] with the Naïve Acceptance [NA] criterion.
   (b) Time Weighted Phase Selector (1) [T1] with Simulated Annealing [SA] acceptance criterion.
2. Generally perform poorly: Any operator selection vector with:
   (a) The Exponential Monte Carlo acceptance criterion [MC].
   (b) The Improving or Equal Only acceptance criterion [IO].

Correctly setting the early termination criteria means the hyper-heuristic can determine the correct computational time even though it has no knowledge of the problem instance size or features. We propose to undertake further work on this feature.

In future research we shall examine whether the relative performance of the operator selection vector and acceptance criterion pairings is consistent across problem instances of differing sizes. We shall also evaluate the merits of enabling the adaptive hyper-heuristic to change the pairing of operator selection vector and acceptance criteria during an interim phase update.

References

1. Bai, R., Blazewicz, J., Burke, E., Kendall, G., McCollum, B.: A simulated annealing hyper-heuristic methodology for flexible decision support. 4OR Q. J. Oper. Res. **10**(1), 43–66 (2012)
2. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Burke, E., Erben, W. (eds.) Practice and Theory of Automated Timetabling III. LNCS, vol. 2079. Springer, Heidelberg (2000)
3. Dueck, G.: New optimization heuristics: the great deluge algorithm and the record-to-record travel. J. Comput. Phys. **104**(1), 86–92 (1993)
4. Fisher, M.: Optimal solution of vehicle routing problems using minimum k-trees. Oper. Res. **42**(4), 626–642 (1994)
5. Glover, F.: Tabu search: Part I. ORSA J. Comput. **1**(3), 190–206 (1989)
6. Marshall, R.J., Johnston, M., Zhang, M.: A comparison between two evolutionary hyper-heuristics for combinatorial optimisation. In: Dick, G., Browne, W.N., Whigham, P., Zhang, M., Bui, L.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K.C., Tang, K. (eds.) SEAL 2014. LNCS, vol. 8886, pp. 618–630. Springer, Heidelberg (2014)
7. Misir, M., Verbeeck, K., De Causmaecker, P.: A new hyper-heuristic as a general problem solver: an implementation in HyFlex. J. Sched. **16**, 291–311 (2013)
8. Ochoa, G., Hyde, M.: Cross-domain Heuristic Search Challenge (2011). http://www.asap.cs.nott.ac.uk/external/chesc2011/
9. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K.: HyFlex: a benchmark framework for cross-domain heuristic search. In: Hao, J.-K., Middendorf, M. (eds.) EvoCOP 2012. LNCS, vol. 7245, pp. 136–147. Springer, Heidelberg (2012)
10. Sabar, N., Ayob, M., Kendall, G., Qu, R.: Grammatical evolution hyper-heuristic for combinatorial optimization problems. IEEE Trans. Evol. Comput. **17**(6), 840–861 (2013)

11. Soubeiga, E.: Development and application of hyperheuristics to personnel scheduling. Ph.D thesis, University of Nottingham (2003)
12. Toth, P., Vigo, D.: The Vehicle Routing Problem. SIAM. LNCS. Springer, Philadelphia (2002)
13. Walker, J.D., Ochoa, G., Gendreau, M., Burke, E.K.: Vehicle routing and adaptive iterated local search within the *HyFlex* hyper-heuristic framework. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, vol. 7219, pp. 265–276. Springer, Heidelberg (2012)