# A Case Study of Controlling Crossover in a Selection Hyper-heuristic Framework Using the Multidimensional Knapsack Problem

**John H. Drake** drakejohnh@gmail.com
School of Computer Science, University of Nottingham,
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK

**Ender Özcan** ender.ozcan@nottingham.ac.uk
School of Computer Science, University of Nottingham,
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK

**Edmund K. Burke** e.k.burke@stir.ac.uk
Computing Science and Mathematics, School of Natural Sciences,
University of Stirling, Stirling, FK9 4LA, Scotland

doi:10.1162/EVCO_a_00145

**Abstract**

Hyper-heuristics are high-level methodologies for solving complex problems that operate on a search space of heuristics. In a selection hyper-heuristic framework, a heuristic is chosen from an existing set of low-level heuristics and applied to the current solution to produce a new solution at each point in the search. The use of crossover low-level heuristics is possible in an increasing number of general-purpose hyper-heuristic tools such as HyFlex and Hyperion. However, little work has been undertaken to assess how best to utilise it. Since a single-point search hyper-heuristic operates on a single candidate solution, and two candidate solutions are required for crossover, a mechanism is required to control the choice of the other solution. The frameworks we propose maintain a list of potential solutions for use in crossover. We investigate the use of such lists at two conceptual levels. First, crossover is controlled at the hyper-heuristic level where no problem-specific information is required. Second, it is controlled at the problem domain level where problem-specific information is used to produce good-quality solutions to use in crossover. A number of selection hyper-heuristics are compared using these frameworks over three benchmark libraries with varying properties for an NP-hard optimisation problem: the multidimensional 0-1 knapsack problem. It is shown that allowing crossover to be managed at the domain level outperforms managing crossover at the hyper-heuristic level in this problem domain.

**Keywords**

Combinatorial optimisation, hyper-heuristics, local search, multidimensional knapsack problem, metaheuristic.

## 1 Introduction

Hyper-heuristics are high-level search methodologies used to solve computationally difficult problems. Unlike traditional techniques, a hyper-heuristic operates on a search

Manuscript received: December 9, 2012; revised: October 14, 2013, February 24, 2014, and September 26, 2014; accepted: December 22, 2014.

space of low-level heuristics rather than directly on the search space of solutions. A recent definition of hyper-heuristics is offered by Burke et al. (2010, 2013):

A hyper-heuristic is a search method or learning mechanism for selecting or generating heuristics to solve computational search problems.

This terminology includes systems covering the two main classes of hyper-heuristics, those concerned with heuristic *selection* and those with heuristic *generation*. Here we work with selection hyper-heuristics. Operating on a single solution, low-level heuristics are repeatedly selected and applied, with a decision made as to whether to accept the move until some termination criterion is met. Hyper-heuristics have been applied to a wide range of problems such as examination timetabling (Özcan et al., 2009; Burke, Kendall et al., 2003), production scheduling (Fisher and Thompson, 1961), nurse scheduling (Burke, Kendall et al., 2003), and vehicle routing (Drake et al., 2013).

Crossover is often used in population-based metaheuristics as a mechanism to recombine multiple solutions. This causes a problem in single-point search because each operator requires two solutions as input. In this paper, we investigate the management of input arguments for crossover operators in single-point search hyper-heuristics. We define frameworks at two conceptual levels to control crossover in single-point hyper-heuristics. Experiments are performed to analyse the performance difference between allowing a hyper-heuristic to select the second argument for a binary crossover operator using domain-independent knowledge, or controlling this decision directly in the problem domain using domain-specific knowledge. Our hyper-heuristics are applied to three benchmark libraries for the multidimensional 0-1 knapsack problem (MKP), each with varying properties, something that has not been done in any previous studies. We are not trying to achieve state-of-the-art results in this domain; our focus is to use the MKP as a benchmark to compare the proposed frameworks for crossover control.

Section 2 provides an overview of hyper-heuristics; a definition and brief classification of hyper-heuristic methods is included. This is followed by more detailed discussion of selection hyper-heuristics and the use of crossover within this paradigm. Section 3 gives an overview of the MKP literature. Section 4 introduces crossover management at two different levels, provides detailed information on the selection hyper-heuristics used in this paper, and defines the MKP benchmarks used as a test bed. Section 5 details some preliminary experimentation performed to decide the initialisation method to use in domain-specific crossover management. Section 6 provides results and discussion of the proposed selection hyper-heuristics applied to the MKP. Finally, Section 7 draws some conclusions based on our results.

## 2 Hyper-heuristics

There are currently two main categories of hyper-heuristics, as outlined by Burke et al. (2010). The first category contains those methodologies that select low-level heuristics to apply (from a set of existing heuristics). The second contains methodologies that create new heuristics from a set of heuristic components of other existing low-level heuristics (Burke, Hyde et al., 2009). These categories are then further broken down to distinguish between hyper-heuristics that construct solutions from scratch (Burke et al., 2007) and those that aim to improve existing complete solutions (Özcan et al., 2009). Aside from the nature of the search space, many hyper-heuristics learn from feedback given regarding heuristic performance to guide low-level heuristic choice. Such feedback is used to learn in either an online or an offline manner. Online learning occurs during the process of solving a problem instance (Drake et al., 2012). In offline learning a system is trained on a subset of problems prior to full execution in order to assert a set of rules to apply to unseen instances (Hyde, 2010).

Burke et al. (2013) identified a number of closely related areas to hyper-heuristic research, including: adaptive operator selection (Fialho et al., 2008), reactive search (Battiti et al., 2008), variable neighborhood search (Nenad and Pierre, 1997), adaptive memetic algorithms (Ong et al., 2006), and algorithm portfolios (Huberman et al., 1997). Although an overview of these methods is not provided here, a number of approaches discussed overlap these areas. The references provided are a good starting point for learning about each of these techniques.

## 2.1 Single-Point Search Hyper-heuristics

The traditional single-point search hyper-heuristic framework relies on two key components, a heuristic selection method and a move acceptance criterion. Such hyper-heuristics are labelled *selection method–acceptance criterion* in this article.

Cowling et al. (2001a) experimented with a number of heuristic selection methods including simple random and choice function using two simple move acceptance criteria, accept all moves and accept only improving moves. In this early work, choice function heuristic selection combined with the all moves acceptance criterion was shown to work well. The choice function was used as a selection mechanism in a number of further studies (Bilgin et al., 2006; Kiraz et al., 2013).

Nareyek (2001) analysed a number of weight adaptation functions and two mechanisms to choose heuristics within a reinforcement learning (Sutton and Barto, 1998) heuristic selection method. Taking the low-level heuristic with the maximum utility value, rather than using a weighted probability of selection and using additive and subtractive weight adaptation, was shown to be a reasonable choice when using reinforcement learning.

Özcan et al. (2009) used late acceptance strategy hill climbing as a move acceptance criterion within a single-point search hyper-heuristic framework to solve standard benchmarks of the examination timetabling problem. This work suggested that the late acceptance strategy was relatively successful when used with simple random heuristic selection and less suitable when used with more intelligent methods such as choice function and reinforcement learning. Late acceptance strategy-based hyper-heuristics were also explored by Jackson et al. (2013) in the context of cross-domain optimisation.

García-Villoria et al. (2011) applied a number of different hyper-heuristic methods to an NP-hard scheduling problem, the response time variability problem. After introducing a number of constructive hyper-heuristics for this problem, several single-point search hyper-heuristics were tested. Simple random, greedy, and two probability-based heuristic selection methods were used to select a heuristic from a set of local search operators with all moves accepted. Using a hyper-heuristic to select a local search heuristic was shown to outperform naive iterative selection. The local search heuristics were then replaced by a set of metaheuristics. The combination of metaheuristics within a hyper-heuristic framework was observed to be superior to applying each of the metaheuristics individually.

Burke et al. (2012) applied a number of hyper-heuristics to a set of examination timetabling instances. Hyper-heuristics using either simple random, greedy, choice function, or reinforcement learning heuristic selection were tested in combination with three move acceptance criteria based on simulated annealing. The hyper-heuristics utilising reinforcement learning performed poorly in these studies. Better performance was observed when using simple random selection with the same move acceptance criterion. That an "intelligent" mechanism is unable to learn which heuristic to apply at a given time suggests a complex relationship between heuristic selection methods and move acceptance criteria.

Demeester et al. (2012) used simple random hyper-heuristics to solve three exam timetabling datasets. Improving or equal, great deluge, simulated annealing, late acceptance strategy, and steepest descent late acceptance strategy were used as move acceptance criteria. The simple random–simulated annealing hyper-heuristic improved on a number of best results from the literature over the Toronto benchmark dataset. This hyper-heuristic also performed well over a second dataset provided by the authors. Other hyper-heuristics using the simulated annealing move acceptance criterion have been applied to a number of domains, including the multimodal homecare scheduling problem (Rendl et al., 2012), bin packing (Bai et al., 2012), and university course timetabling (Bai et al., 2012).

The work in this paper uses many single-point search hyper-heuristics such as those described here. A large number of other heuristic selection methods and move acceptance criteria exist in the literature. A complete description of all these mechanisms is beyond the scope of this paper, but a number of survey papers (Burke et al., 2013; Ross, 2005; Chakhlevitch and Cowling, 2008) provide a thorough grounding in this area.

## 2.2 Hyper-heuristic Frameworks

Özcan et al. (2008) describe and compare four different hyper-heuristic frameworks operating over a set of perturbative low-level heuristics. Perturbative heuristics can be split into two categories, *mutational heuristics* and *hill climbers*. A mutational heuristic takes a solution as input, performs an operation to perturb the solution, and outputs a new solution without quality guarantee. A hill climber accepts a solution as input, performs an operation to perturb the solution, and guarantees to output a solution whose quality is at least as good as the original input.

Of these frameworks, $F_A$ is the traditional hyper-heuristic framework where a low-level heuristic is selected and applied, with the resulting solution subsequently accepted or rejected based on the quality of the new solution. $F_B$ selects a low-level heuristic from a set of mutational heuristics and hill climbers. If a mutational heuristic is selected, a hill climber from the available set is then also applied before a decision whether to accept or reject the move is made. $F_C$ selects and applies a mutational heuristic $LLH_i \in LLH_1, \ldots, LLH_n$, where $n$ is the number of mutational heuristics available, followed by a predefined hill climber $HC$ before deciding whether to accept the new solution. Such a framework is illustrated in Figure 1 and is the framework used in this paper. $F_D$ distinctly separates mutational heuristics and hill climbers into two groups. A mutational heuristic is chosen and applied from the first group and accepted or rejected based on performance. A hill climber from the second group is then applied, and a separate decision is made whether to accept or reject the move. $F_C$ was found to yield better results than the traditional $F_A$ framework on a number of benchmark functions.

Hyper-heuristics operating using an $F_C$ framework have similar characteristics to memetic algorithms. Memetic algorithms (Moscato et al., 2004) combine evolutionary algorithms and local search techniques. A simple memetic algorithm will attempt to improve each candidate solution in a population through some hill climbing

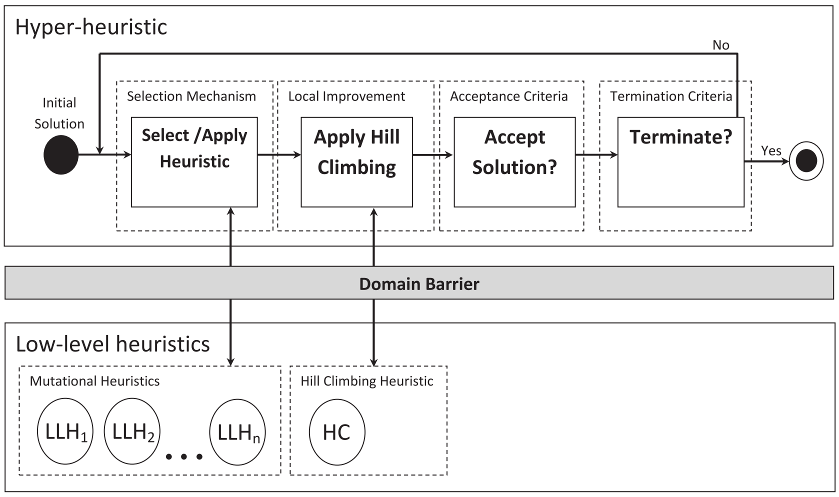

Figure 1: Single-point search hyper-heuristic framework with local improvement ($F_C$).

mechanism. Memetic algorithms have been demonstrated to be successful over a number of different problem domains, including the MKP (Chu and Beasley, 1998; Özcan and Basaran, 2009). In this paper, we analyse the effect of introducing crossover low-level heuristics into an $F_C$ hyper-heuristic framework.

## 2.3 Crossover in Single-Point Search Hyper-heuristics

Evolutionary algorithms (EAs) maintain a population of individuals through the processes of selection, crossover, and mutation. Given two or more suitably fit solutions (parents), the underlying principle of crossover is to recombine them to produce new solutions (children) in such a way that the child solutions inherit the good characteristics of all parents. Each generation of the population consists of the best solutions from the previous generation on the newly generated child solutions. The intention of this process is to gradually improve the quality of the individuals in the population. A large number of crossover operators are proposed in the literature for general and specific purposes.

Despite the support for crossover operators in modern hyper-heuristic frameworks such as HyFlex (Burke, Curtois et al., 2009), Hyperion (Swan et al., 2011), and ParHyFlex (Onsem and Demoen, 2013), limited research effort has been directed at managing this type of low-level heuristic. In a recent competition based on the HyFlex framework (Ochoa and Hyde, 2011), only two of the top ten entrants provided a description of a strategy to control candidate solutions to use as the input arguments for crossover low-level heuristics.

Of these two, one simply uses the current solution as the first candidate and the best seen solution so far as the second candidate. The other provides a detailed explanation of a crossover management scheme and was the eventual competition winner (Misir et al., 2012). This hyper-heuristic also uses the current solution as one candidate and maintains a memory of the five best solutions seen so far to use as second candidate solutions. Each time a crossover low-level heuristic is selected, a random solution from this memory is used. When a new best-of-run solution is found, it replaces one of the five solutions in memory chosen at random. More recently Kheiri and Özcan (2013) used a simple scheme to manage solutions to use as second candidate solutions for crossover low-level heuristics, again using the HyFlex framework. A circular list containing the best solutions seen so far is maintained; however, the length of the list is arbitrarily set. A pointer indicates which solution is to be used each time a crossover low-level heuristic requires a second solution and is advanced to the next solution in the list after each application of crossover.

These methods relate to hyper-heuristics managing the input solutions for crossover low-level heuristics at the hyper-heuristic level. Cobos et al. (2011) presented two selection hyper-heuristics operating over a set of metaheuristics, including genetic algorithm variants. Rather than a single-point search framework, the low-level heuristics in this framework operate over a shared population of solutions. The genetic algorithm variants perform crossover on two individuals selected from this shared population. In this case, the responsibility for providing the two candidate solutions necessary for crossover is below the domain barrier and is managed by the low-level heuristics rather than at the hyper-heuristic level.

Maturana et al. (2010) selected a crossover low-level heuristic to use at each step of evolutionary algorithms for the satisfiability problem (SAT). Although the choice of heuristic is made at the hyper-heuristic level, the selection of input solutions is performed at the domain level. Using two-parent crossover for all of the crossover heuristics available, the candidate solutions are selected using two schemes. In the early experimentation, this selection is performed randomly between all solutions in the population. A fitness-biased selection scheme is also used; however, the details of this mechanism are not explained.

The management of the candidate solutions required for crossover in selection hyper-heuristics is often overlooked by many researchers. Indeed there are no standard mechanisms defined for controlling crossover in this context, nor is there any mention of crossover management in any of the survey papers mentioned in Section 2.1. An open research question is whether the responsibility of providing input for crossover (and other multiargument low-level heuristics) in selection hyper-heuristics should lie with the high-level hyper-heuristic or with the low-level heuristics operating below the domain barrier. This is particularly important if it is considered that managing these solutions at the hyper-heuristic level is in breach of crossing the domain barrier.

## 3 The Multidimensional Knapsack Problem

The NP-hard (Garey and Johnson, 1979) multidimensional 0-1 knapsack problem (Weingartner and Ness, 1967) is a generalised case of the 0-1 knapsack problem whose roots can be traced back to capital budgeting and project selection applications. The MKP is a resource allocation model, with the objective of selecting a subset of objects yielding the greatest profit while observing the constraints on knapsack capacities. Each object $j$ consumes a different amount of resources in each dimension $i$ when selected. Formally the MKP can be stated as

$$\text{maximise} \quad \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m \tag{2}$$

$$\text{with} \quad x_j \in \{0, 1\}, \quad j = 1, \ldots, n \tag{3}$$

where $p_j$ is the profit for selecting item $j$, $a_{ij}$ is the resource consumption of item $j$ in dimension $i$, and $b_i$ is the capacity constraint of each dimension $i$. With direct binary encoding, $x_1, \ldots, x_n$ is a set of decision variables indicating whether object $j$ is included in the knapsack. The size of a problem is defined by the total number of variables $n$ and the number of dimensions $m$. Tavares et al. (2008) investigated five different representations and analysed their effects on solution quality. This work highlighted that using direct binary encoding in conjunction with local search or repair operators in both mutation-based and crossover-based evolutionary algorithms is suitable for the MKP.

A number of methods, both exact and metaheuristic, have been used to solve the MKP. These include memetic algorithms (Chu and Beasley, 1998; Özcan and Basaran, 2009), tabu search (Vasquez and Hao, 2001), simulated annealing (Qian and Ding, 2007), particle swarm optimisation (Hembecker et al., 2007), kernel search (Angelelli et al., 2010), core-based and tree search algorithms (Mansini and Speranza, 2012; Boussier et al., 2010; Vimont et al., 2008), and genetic algorithms (Khuri et al., 1994). No previous known work uses selection hyper-heuristics to solve the MKP.

The MKP has become a favoured domain for research into hybrid metaheuristics and mathematical programming methods. Such techniques belong to the emerging research field of *matheuristics* (Maniezzo et al., 2010; Raidl and Puchinger, 2008). Matheuristics have been applied to a variety of problem domains, including the MKP (Chu and Beasley, 1998; Puchinger et al., 2006; Raidl, 1998; Vasquez and Vimont, 2005; Hanafi et al., 2010; Croce and Grosso, 2012; Hanafi and Wilbaut, 2011), providing some of the best results in the literature. The linear programming (LP) relaxation of the MKP allows the variables $x_j$ from Equation (3) to take fractional values rather than being restricted to discrete values of 0 and 1 as shown in Equation (4):

$$0 \leq x_j \leq 1, \quad j = 1, \ldots, n \tag{4}$$

The LP-relaxed version of the problem is solvable in polynomial time and can provide useful information about the current problem instance. Indeed, some of the best results in the literature are from methods combining LP relaxation and heuristics (Chu and Beasley, 1998; Vasquez and Vimont, 2005). Chu and Beasley (1998) combined a traditional genetic algorithm with a repair operator based on the dual variables of the LP-relaxed problem. Raidl (1998) used a similar method, which considered the actual values of the LP-relaxed solution when repairing candidate solutions. Puchinger et al. (2006) explored the core concept for the MKP. The core concept reduces the problem to a subset of decision variables that presents the most difficult decision as to whether they are in an optimal solution. The core concept fixes the variables of high and low efficiency and restricts the optimisation to the difficult-to-place medium efficiency items. A memetic algorithm and guided variable neighbourhood search are also implemented on the restricted version of the problem, showing better results than when applied to the original problem directly. Vasquez and Vimont (2005) obtained the best known results for the largest instances from the benchmarks of Chu and Beasley (1998). Their approach applied tabu search to promising areas of the search space derived from LP-relaxed optima with an improved algorithm fixing additional variables matching the attributes of a good solution.

The multidimensional knapsack problem has been chosen as a test bed for two reasons. First, because the MKP can be represented as a binary bit string, a large number of general low-level heuristics already exist in the literature. Second, a large number of different benchmark datasets exist for this problem. The availability of these

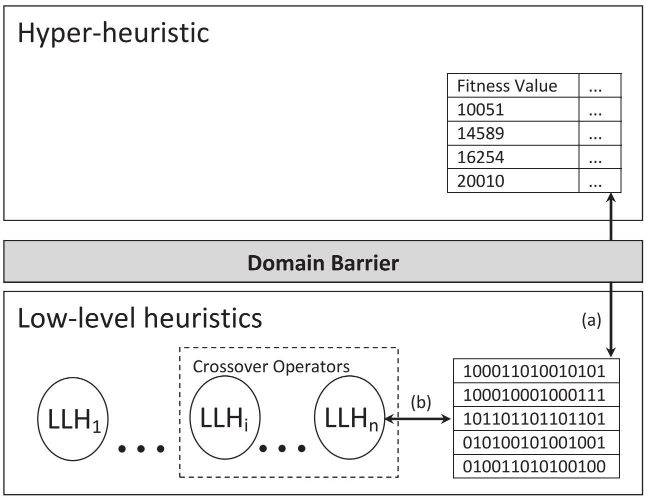

Figure 2: A general framework for controlling crossover; hyper-heuristic control is shown by arrow a and low-level control by arrow b.

benchmarks allows us to test the generality of the methods we use over a wide variety of problem instance types within a single problem domain.

## 4 Controlling Crossover in Selection Hyper-heuristics for the Multidimensional Knapsack Problem

Traditionally crossover is included in population-based approaches, as opposed to the single-point search used in many selection hyper-heuristics. In binary crossover two candidate solutions are selected from a population and a new solution is generated containing material from both parents. In the case of single-point search hyper-heuristics, a trivial selection for one of the candidate solutions is the current solution. The questions of where the second candidate solution should come from and at which level it should be managed are more difficult to answer. It is not obvious at which level the second candidate solution for crossover should be managed, so we propose two frameworks. In each case, a list of potential solutions for crossover is maintained. The general shared framework is shown in Figure 2, with a set of crossover low-level heuristics $LLH_i, \ldots, LLH_n$ operating on set of candidate solutions represented as binary strings.

The first framework maintains a list at the hyper-heuristic level. Although the candidate solutions exist below the domain barrier, the hyper-heuristic chooses a solution to use for crossover based on feedback given during the search. This raises a number of questions: What information should be passed back to the hyper-heuristic? How should this list be maintained? How long should this list be? The interaction between the hyper-heuristic and the solutions is depicted by arrow a in Figure 2.

The second framework allows the low-level heuristics to manage the list of second solutions for crossover directly. Again this poses similar questions regarding the size of such a list and how it should be initialised and maintained. This framework is also shown in Figure 2, with the interaction between the low-level heuristics and the solutions depicted by arrow b. Figure 2 should be viewed as an extension to the $F_C$ framework presented in Figure 1.

### 4.1 Controlling Crossover at the Hyper-heuristic Level

A list of candidate solutions to use as input arguments for crossover low-level heuristics can be controlled at the hyper-heuristic level. In *memory crossover*, a list of solutions (that were the best-of-run when found) is maintained, with one of these solutions used each time a crossover operator is selected. Initially this list is populated randomly. Each time a new best-of-run solution is found, it replaces the worst existing solution in the list. This method is similar to the crossover control strategies used by Misir et al. (2012) and Kheiri and Özcan (2013). A method of choosing a solution to use from this memory is needed. Any evolutionary algorithm parent selection method can be used for this; we preferred using *tournament selection*. In tournament selection, a subset of solutions of a given *tournament size* is chosen from a list. These solutions are paired up, the highest-quality solution in a pair is kept, and the other is discarded. The pairing process continues until a single solution is left. This method of crossover control is similar to steady state genetic algorithms, which select and update a population in much the same way.

In order to see if any benefit is gained by controlling crossover in this way, two other methods of choosing the second parent are also tested. *Random crossover*, also known as headless chicken crossover (Jones, 1995), takes the two solutions to be the current solution in the hyper-heuristic and a randomly generated solution. This does not fit in with the original ethos of crossover, which is to preserve and exploit the good characteristics of high-quality solutions. In addition, each hyper-heuristic is also tested with crossover low-level heuristics omitted completely.

### 4.2 Controlling Crossover at the Domain-Specific Level

It is also possible to maintain a list of candidate solutions at the domain-specific level. Here, problem-specific heuristics are used to populate a static list of candidate solutions generated based on problem domain-specific knowledge. One of these solutions is then used as the second candidate solution during a crossover operation. The list is static, since we expect the solutions in the list to contain the building blocks of high-quality solutions. This is implemented as a queue of solutions whereby each time a solution is required for crossover, the solution at the head of the queue is taken. This solution will be used in the crossover operation before being placed at the tail of the queue. Some procedure must be defined to initialise this list.

A number of methods exist in the literature to initialise solutions for the MKP. Gottlieb (2000) compared a number of initialisation methods for evolutionary algorithms solving the MKP. The two best initialisation routines of this study were C* and R*. C* is a variation of the method of Chu and Beasley (1998) whereby starting with an empty solution, the algorithm attempts to add each item in a random order. R* is based on a method originally proposed by Raidl (1998) and uses the solutions to the LP-relaxed version of each problem to construct each candidate solution. A potential drawback of both of these approaches is that as only feasible solutions can be generated, there are a large number of infeasible solutions close to optimal solutions not considered.

A new initialisation method allowing infeasible solutions *jqdInit* is proposed. This method is shown in Algorithm 1. Given a solution $S \in \{0, 1\}^n$ starting with no items selected, each item $j$ is considered sequentially. An item is included in the solution with probability equal to its value in the LP-relaxed solution irrespective of whether a feasible solution is obtained. Pseudorandom numbers $R_j$ $(0 \leq R_j < 1)$ are used in this step. In terms of time complexity, all three initialisation methods must visit every variable in $n$ once and so are asymptotically equivalent running in $O(n)$ time.

**Algorithm 1:** Generating MKP solutions allowing infeasibility ($jqdInit$)

```
1: Let $x_k^{LP}$ represent the LP-relaxed solution of item $k$
2: Set $S_j \leftarrow 0, \forall j \in 1, ..., n$
3: for $j = 1$ to $n$ do
4:    if $x_j^{LP} \geq R_j$ then
5:       $S_j \leftarrow 1$
6:    end if
7: end for
8: return $S$
```

## 4.3 Hyper-heuristic Components

The heuristic selection methods and move acceptance criteria used in this paper are introduced in Sections 4.3.1 and 4.3.2.

### 4.3.1 Selection Mechanisms

*4.3.1.1 Simple Random (SR).* This randomly selects a heuristic from the set of low-level heuristics at each point in the search.

*4.3.1.2 Choice Function (CF).* This is a more elegant selection method that scores heuristics based on a combination of three different measures before applying the heuristic with the highest rank. The first measure ($f_1$) records the previous performance of each individual heuristic, with more recent executions carrying larger weight. The value of $f_1$ for each low-level heuristic $h_1, h_2, \ldots, h_j$ is calculated as

$$f_1(h_j) = \sum_n \alpha^{n-1} \frac{I_n(h_j)}{T_n(h_j)}, \tag{5}$$

where $I_n(h_j)$ is the change in evaluation function and $T_n(h_j)$ is the time spent calling the heuristic for each previous invocation $n$ of heuristic $h_j$. $\alpha$ is a value between 0 and 1 giving greater importance to recent performance.

The second ($f_2$) measures previous performance following the last low-level heuristic chosen, in an attempt to capture any pairwise dependencies between heuristics. Values for $f_2$ are calculated in a similar fashion for each heuristic $h_j$ when invoked immediately following $h_k$, as shown in Equation (6):

$$f_2(h_j, h_k) = \sum_n \beta^{n-1} \frac{I_n(h_k, h_j)}{T_n(h_k, h_j)}, \tag{6}$$

where $I_n(h_k, h_j)$ is the change in evaluation function and $T_n(h_k, h_j)$ is the time spent calling the heuristic for each previous invocation $n$ of heuristic $h_j$ following $h_k$. $\beta$ is a value between 0 and 1 giving greater importance to recent performance.

The final measure ($f_3$) is simply the time elapsed ($\tau(h_j)$) since the heuristic was last executed, included to add an element of diversity to the low-level heuristics chosen:

$$f_3(h_j) = \tau(h_j). \tag{7}$$

A score for each heuristic is given in order to rank heuristics, calculated as choice function $F$:

$$F(h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \delta f_3(h_j), \tag{8}$$

where the previously defined $\alpha$ and $\beta$ weight $f_1$ and $f_2$, respectively, to provide sufficient intensification of the search process, and $\delta$ weights $f_3$ to provide sufficient diversification. In this paper, the values of $\alpha$, $\beta$, and $\delta$ are controlled using the parameter-free scheme of Cowling et al. (2001b).

*4.3.1.3 Reinforcement Learning (RL).* This assigns a utility weight to each low-level heuristic. If a heuristic improves a solution, this weight is increased by an amount defined by the chosen adaptation function. If a heuristic does not improve a solution, this weight is decreased accordingly. Heuristic selection at the next step of the search is then based on these values, choosing randomly between the heuristics with the largest utility weight.

### 4.3.2 Move Acceptance Criteria

*4.3.2.1 Only Improving (OI).* This is a simple move acceptance criterion whereby any improving move made by application of a low-level heuristic chosen by the selection method is accepted.

*4.3.2.2 Simulated Annealing (SA).* This is a generic metaheuristic technique for optimisation often used as an acceptance criterion in hyper-heuristics (Kirkpatrick et al., 1983). In simulated annealing a move resulting in a solution of equal or greater quality than the previous move is accepted. If a move yields a solution of poorer quality, the move is accepted probabilistically based on the decrease in solution quality and a temperature parameter that decreases over time. The probability of moving to a worse solution will reduce over time as the temperature decreases. This probability $p$ is given as

$$p = \frac{1}{1 + e^{-\Delta/T}}, \tag{9}$$

where $\Delta$ is the change in fitness function value and $T$ is the current temperature value.

*4.3.2.3 Late Acceptance Strategy (LAS).* This promotes a general trend of improvement throughout a search process, comparing a candidate solution to one generated a specified number of steps before kept in memory (Burke and Bykov, 2008). If the current solution is better than the previous solution in memory, it replaces that solution and the next oldest solution is used for the next comparison. If the current solution is worse than the old solution, the last solution accepted replaces the old solution.

## 4.4 Low-Level Heuristics

Several standard low-level heuristics from the literature have been implemented. In the case of crossover low-level heuristics, two children are generated each time a low-level heuristic of this type is selected, with the best solution kept for consideration by the move acceptance criterion.

### 4.4.1 One-Point Crossover (1PX)

Given two candidate solutions, 1PX (Goldberg, 1989) selects a single crossover point at random and exchanges the genetic data that appear on one side of this point between the two solutions.

### 4.4.2 Two-Point Crossover (2PX)

This heuristic (Goldberg, 1989) is similar to 1PX except two crossover sites are given and the genetic material that is contained within these two sites is exchanged.

### 4.4.3 Uniform Crossover (UX)

This considers each position within two chosen candidate solutions and exchanges each bit with a given exchange probability $p_e$, set at .5 (Syswerda, 1989) .

### 4.4.4 Swap Mutation (SWP)

This selects two distinct substrings of a candidate and exchanges their positions to generate a new solution (Özcan et al., 2006). The length of these substrings is set to the number of variables $n/10$.

### 4.4.5 Parameterised Mutation (PARAxx)

This inverts a specified number of bits within a solution. This is essentially the bit string mutation of Koza (1992). However, rather than relying on mutational probabilities, parameterised mutation guarantees the number of bits that are mutated during the operation. In these experiments, three variations of this operator are implemented to perform light, medium, and heavy mutation at rates of 10% (PARA10), 25% (PARA25), and 50% (PARA50), respectively.

### 4.4.6 Hill Climbing Heuristic

A number of papers in the literature (Chu and Beasley, 1998; Pirkul, 1987; Magazine and Oguz, 1984) make use of an *add* and (or) *drop* phase to construct, improve, or repair solutions to the MKP. These techniques more often than not use a *utility weight* value to sort objects in order of their relative efficiency. Chu and Beasley (1998) adopt the *surrogate duality* suggested by Pirkul (1987), multiplying each weight by a relevance value $r$:

$$util_j = \frac{p_j}{\sum_{i=1}^{m} r_i w_{ij}}. \tag{10}$$

Relevance values $r_i$ are taken to be the *dual variables* of each dimension $i$ in the solution to the LP relaxation of the MKP. By use of these relevance values a local search operator for the MKP can be implemented. When given an infeasible solution, *drop* items from the knapsack in order of increasing *utility weight* until a feasible solution is found. When a feasible solution is obtained, attempt to *add* items in order of decreasing *utility weight* until a feasible solution cannot be found by adding another of the unselected items. Puchinger et al. (2006) tested a number of efficiency methods, with the relevance weights of Chu and Beasley (1998) observed to be the best efficiency measure for the MKP. In our experiments, this operator is applied as a local search mechanism after each crossover or mutational operator is applied to repair and locally improve solutions, as required by the $F_C$ selection hyper-heuristic framework.

## 4.5 Experimental Data and Test Framework Definitions

There are three well-known benchmark libraries for the MKP in the literature. SAC-94 is a standard set of MKP instances from a number of papers in the literature, often representing real-world examples. These instances are generally small, with $m$ ranging from 2 to 30, $n$ ranging from 10 to 105, and optimal solutions known for all. Chu and Beasley (1998) noted that the SAC-94 instances are too small to yield meaningful conclusions of an algorithm's performance, leading to the proposal of the ORLib instances. This is a widely used benchmark library in the literature and contains 270 instances containing $n \in \{100, 250, 500\}$ variables, $m \in \{5, 10, 30\}$ dimensions, and *tightness ratio* $\in \{0.25, 0.50, 0.75\}$. As optimal solutions are unknown for some of these instances, performance is measured using the %-gap distance from the upper bound provided by the solution to the LP-relaxed problem, calculated as

$$100 * \frac{LPopt - SolutionFound}{LPopt} \tag{11}$$

A third benchmark set was provided by Glover and Kochenberger (2000), referred to here as GK, including much larger instances with $n$ between 100 and 2,500 and $m$ between 15 and 100. Again, optimal solutions are not known for all instances, so performance is measured in terms of %-gap. All instances are available in a unified format at http://doi.org/10.13140/2.1.3578.9122

A run terminates after $10^6$ fitness evaluations for each problem instance in order to directly compare results with the techniques in the literature (Chu and Beasley, 1998; Özcan and Basaran, 2009). Initial solutions are set as a single random binary string of length $n$, where $n$ is the total number of objects associated with each instance. For tests using the SAC-94 benchmark set, a single run of each hyper-heuristic is sufficient, as these instances are extremely small. In the case of the OR-Lib benchmark, each set of 10 instances is taken from same distribution. As a result, taking the average %-gap over these 10 instances for each of the 27 sets effectively shows the performance of 10 runs of each hyper-heuristic. For the larger GK instances, each of the experiments are repeated 10 times to account for the stochastic nature of the hyper-heuristics, with average performance over 10 runs reported. A list length of 500 is used in the late acceptance strategy–based hyper-heuristics, as suggested by previous approaches (Burke and Bykov, 2008; Özcan et al., 2009). Simulated annealing calculates the probability $p$ of accepting a solution (see Section 4.3.2). The initial value of $T$ is set to the difference between the initial solution and the solution obtained by solving the LP-relaxed version of the problem. During the search process $T$ is reduced to 0 in a linear fashion proportional to the number of fitness evaluations left. In the hyper-heuristics using reinforcement learning heuristic selection, the parameters are derived from Nareyek (2001). The utility values for each low-level heuristic are initially set to 10. In each case the application of a low-level heuristic leads to an improvement in the quality of solution; the utility value for this heuristic is incremented by 1, otherwise it is decreased by 1. The utility value of an individual low-level heuristic is bound by a maximum value of 30 and a minimum value of 0. In all experiments using memory crossover, a memory size of $0.1 * n$ is used, where $n$ is the number of variables in the instance currently being solved. This parameter setting is based on a set of preliminary experiments to determine the best value $c$ value for $c * n$, where $c \in \{0.1, 0.2, \ldots, 1.0\}$. These results showed that a small value for $c$ is preferable. This ensures that poor-quality solutions found early in the search are removed from the list quickly, in favour of better-quality solutions found later. Solutions are selected from the memory using tournament selection with a tournament size of 2. All hyper-heuristic experiments were carried out on an Intel Core 2 Duo 3 GHz CPU with 2 GB memory.

## 4.6 Fitness Function

A measure is needed to assess the quality of each solution. There are a number of options when choosing a fitness function for the MKP. In this work, the following fitness function from Özcan and Basaran (2009) is used:

$$profit - o * s * (maxProfit + 1), \tag{12}$$

where *profit* is the profit gained from the items currently selected for inclusion, $o$ is the number of overfilled knapsacks, $s$ is the number of selected items, and *maxProfit* is the largest profit value of any of the items. This fitness function will always be positive for a feasible solution and negative for an infeasible solution.

Table 1: Average best solutions for C*, R*, and *jqdInit* initialisation methods over each set of 10 instances in the 90 ORLib instances with $m = 5$.

| Instance set | C* | R* | *jqdInit* |
|---|---|---|---|
| OR5x100-0.25 | $19105_{2.31}$ | $23948_{0.37}$ | $16325_{0.57}$ |
| OR5x100-0.50 | $37136_{1.91}$ | $43015_{0.26}$ | $42742_{0.69}$ |
| OR5x100-0.75 | $55909_{0.86}$ | $60158_{0.23}$ | $60082_{0.33}$ |
| OR5x250-0.25 | $47840_{1.19}$ | $60137_{0.16}$ | $59902_{0.32}$ |
| OR5x250-0.50 | $94016_{0.51}$ | $109080_{0.10}$ | $108653_{0.24}$ |
| OR5x250-0.75 | $140632_{0.44}$ | $151344_{0.06}$ | $151255_{0.10}$ |
| OR5x500-0.25 | $94431_{0.86}$ | $120392_{0.05}$ | $119937_{0.27}$ |
| OR5x500-0.50 | $188748_{0.65}$ | $219323_{0.03}$ | $218962_{0.11}$ |
| OR5x500-0.75 | $280437_{0.35}$ | $302185_{0.02}$ | $301870_{0.05}$ |

Table 2: Average list quality for C*, R*, and *jqdInit* initialisation methods over each set of 10 instances in the 90 ORLib instances with $m = 5$.

| Instance set | C* | R* | *jqdInit* |
|---|---|---|---|
| OR5x100-0.25 | $17781_{1.88}$ | $23545_{0.29}$ | $-64285_{63.77}$ |
| OR5x100-0.50 | $35553_{1.31}$ | $42575_{0.38}$ | $-97229_{65.86}$ |
| OR5x100-0.75 | $54633_{0.75}$ | $59777_{0.22}$ | $-184816_{73.04}$ |
| OR5x250-0.25 | $45045_{1.07}$ | $59739_{0.15}$ | $-181532_{53.70}$ |
| OR5x250-0.50 | $90814_{0.39}$ | $108657_{0.11}$ | $-284952_{73.26}$ |
| OR5x250-0.75 | $137421_{0.32}$ | $150905_{0.08}$ | $-436705_{81.19}$ |
| OR5x500-0.25 | $90016_{0.75}$ | $119951_{0.06}$ | $-348724_{60.51}$ |
| OR5x500-0.50 | $183289_{0.26}$ | $218853_{0.03}$ | $-620003_{52.80}$ |
| OR5x500-0.75 | $275380_{0.19}$ | $301674_{0.01}$ | $-918566_{62.05}$ |

## 5 Finding a Suitable Initialisation Method for the List of Solutions Used in Domain-Level Crossover Control

Some preliminary experiments are required in order to validate the new initialisation method proposed in Section 4.2. The three initialisation techniques described in Section 4.2 (C*, R*, and *jqdInit*) are tested on a subset of 90 instances of ORLib, where $m \in \{5\}$ and $n \in \{100, 250, 500\}$ using a simple random-only improving hyper-heuristic. The hyper-heuristic is allowed to run for $10^6$ fitness evaluations on each instance. In each case 0.1 * $n$ solutions are generated by each initialisation method, where $n$ is the number of variables in the instance currently being solved. Table 1 details the average of the best solution in the list for each initialisation method, and Table 2 shows the average solution quality of all solutions in the list over each set of 10 instances. Standard deviations are given as subscripts. Independent Student's $t$-tests within a 95% confidence interval are performed to assess statistical significance.

The average best solution and average list quality when using R* is far superior to C*. This is unsurprising because R* was designed to generate solutions closer to the optimal than those generated with C*. The best solutions produced by *jqdInit* are also superior to C* on average, with this difference being statistically significant in all cases

Table 3: Performance of initialisation methods over the 90 ORLib instances with $m = 5$.

| Instance set | C* | R* | *jqdInit* |
|---|---|---|---|
| OR5x100-0.25 | 1.31 $_{0.17}$ | 1.48 $_{0.26}$ | 1.25 $_{0.23}$ |
| OR5x100-0.50 | 0.63 $_{0.10}$ | 0.63 $_{0.16}$ | 0.62 $_{0.12}$ |
| OR5x100-0.75 | 0.39 $_{0.07}$ | 0.38 $_{0.11}$ | 0.42 $_{0.08}$ |
| OR5x250-0.25 | 0.70 $_{0.15}$ | 0.51 $_{0.11}$ | 0.45 $_{0.10}$ |
| OR5x250-0.50 | 0.37 $_{0.09}$ | 0.26 $_{0.07}$ | 0.22 $_{0.04}$ |
| OR5x250-0.75 | 0.25 $_{0.06}$ | 0.15 $_{0.04}$ | 0.15 $_{0.04}$ |
| OR5x500-0.25 | 0.70 $_{0.12}$ | 0.25 $_{0.05}$ | 0.24 $_{0.04}$ |
| OR5x500-0.50 | 1.19 $_{0.32}$ | 0.12 $_{0.03}$ | 0.13 $_{0.03}$ |
| OR5x500-0.75 | 0.50 $_{0.18}$ | 0.08 $_{0.02}$ | 0.07 $_{0.01}$ |
| **Average** | 0.67 $_{0.14}$ | 0.43 $_{0.09}$ | 0.39 $_{0.08}$ |

except for OR5x100-0.25. The best solutions produced by *jqdInit* are only slightly poorer quality on average than those produced by R*, with this difference only statistically significant in the case of OR5x100-0.25. As *jqdInit* allows infeasible solutions, the average list quality is very poor in terms of fitness score and of statistically significantly worse quality than both C* and R*. For some instances in this dataset *jqdInit* would not produce any feasible solutions.

Table 3 shows the results obtained in terms of %-gap as the average over 10 instances for each instance set after $10^6$ fitness evaluations. Again, standard deviations are included as subscripts. On these instances C* is the poorest-performing initialisation method, with an average %-gap of 0.67. Using *jqdInit* yields the best results over these instances, achieving an average %-gap of 0.39, slightly outperforming R*, which has an average %-gap of 0.43. Despite both the average and best solutions produced by the R* initialisation being better than the *jqdInit* in all the datasets tested, the new initialisation method leads to better results overall after a full hyper-heuristic run.

The key difference between the existing initialisation methods and the proposed method is the tolerance of infeasible solutions. These solutions may still contain the building blocks of good-quality solutions. The final solution quality does not seem to be adversely affected as a result of this (see Table 3). This suggests that infeasible solutions can help the search process when solving the MKP, particularly as optimal solutions are known to be close to the boundary of feasibility. Because *jqdInit* is competitive with the two existing methods from the literature, it is used during all further experimentation.

## 6 Experiments

Experiments are performed controlling crossover at the hyper-heuristic level and the domain-specific level. In each case, the hyper-heuristics are initially tested over a standard benchmark set before their general applicability is assessed on two further datasets.

### 6.1 Controlling Crossover at the Hyper-heuristic Level for the MKP

As described in Section 4.1, candidate solutions for crossover can be controlled at the hyper-heuristic level with no domain-specific knowledge. When a second individual is required for crossover, it is selected from a list of potential solutions maintained by the hyper-heuristic. To assess the impact of controlling crossover at the hyper-heuristic level in this framework, the experiments are performed for three separate test cases: with

Table 4: Average %-gap over all ORLib instances for each hyper-heuristic with random crossover, memory crossover, and no crossover.

| Hyper-heuristic | Random Crossover | Memory Crossover | No Crossover |
|---|---|---|---|
| SR-OI | 1.16 $_{0.84}$ | 1.12 $_{0.81}$ | 1.11 $_{0.82}$ |
| CF-OI | 1.18 $_{0.83}$ | 1.19 $_{0.86}$ | **1.07** $_{0.80}$ |
| RL-OI | 1.16 $_{0.81}$ | 1.14 $_{0.84}$ | 1.10 $_{0.84}$ |
| SR-LAS | 2.79 $_{2.12}$ | 1.20 $_{0.93}$ | 2.54 $_{1.84}$ |
| CF-LAS | 2.86 $_{2.19}$ | 1.23 $_{0.97}$ | 2.72 $_{2.01}$ |
| RL-LAS | 2.67 $_{1.97}$ | 1.20 $_{0.92}$ | 2.48 $_{1.77}$ |
| SR-SA | 2.35 $_{1.33}$ | 1.21 $_{0.85}$ | 2.10 $_{1.18}$ |
| CF-SA | 2.30 $_{1.29}$ | 1.19 $_{0.82}$ | 2.10 $_{1.19}$ |
| RL-SA | 2.21 $_{1.22}$ | 1.21 $_{0.86}$ | 2.04 $_{1.10}$ |

random crossover, with memory crossover, and with no crossover. Table 4 shows the performance of each hyper-heuristic over all ORLib instances using each of the crossover management strategies (standard deviations are included as subscripts). In this table, the acronyms introduced in Section 4.3 are used for each *selection method–acceptance criterion* combination.

The best performing hyper-heuristic is choice function–only improving with no crossover, with the lowest average %-gap of 1.07 over all ORLib instances. Performing a one-way ANOVA test at a 95% confidence level confirms that there is a statistically significant difference between the performance of the 27 hyper-heuristics. Using the only improving acceptance criterion is clearly superior on average to both late acceptance strategy and simulated annealing in this framework when no crossover or random crossover is used. The results of a post-hoc Tukey's HSD test confirm that these differences are significant, with no statistically significant difference between the techniques sharing a common acceptance criterion. In the case of only improving move acceptance, all three crossover types perform similarly, with no statistically significant difference between results. When using late acceptance strategy and simulated annealing as move acceptance criteria, the performance is significantly better if memory crossover is used. The results obtained using these hyper-heuristics (late acceptance strategy and simulated annealing with memory crossover) do not vary statistically significantly from the hyper-heuristics using the only improving move acceptance criterion.

Overall, the %-gaps of the hyper-heuristics with no crossover are lower than those that use random crossover, suggesting that using crossover as a mutation operator in this way does not benefit the search. This supports previous assertions that the search space of heuristics can be reduced in an attempt to improve performance. Özcan and Basaran (2009) noted that reducing the number of memes can improve the performance of a memetic algorithm solving the MKP. Chakhlevitch and Cowling (2005) also showed similar improvement when reducing the number of low-level heuristics in a hyper-heuristic framework operating on a scheduling problem. For each acceptance criterion there is little difference in the results obtained by using a different heuristic selection method. However, there is significant difference between the results obtained using different move acceptance criteria. This suggests that the acceptance criterion used has a more significant impact on the performance of a hyper-heuristic than the selection mechanism using this heuristic set in this problem domain. This behaviour was also

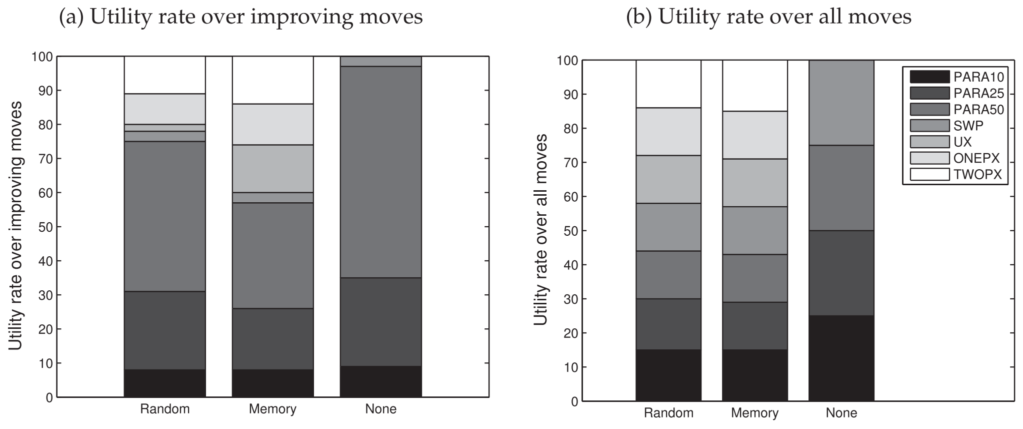

Figure 3: Average low-level heuristic utilisation for choice function–only improving hyper-heuristics with random crossover, memory crossover, and no crossover over all instances in ORLib.

observed by Özcan et al. (2008), where a number of hyper-heuristics were tested over a set of benchmark functions.

Figure 3 shows the utilisation rates of each low-level heuristic for each of the choice function–only improving hyper-heuristics with random crossover, memory crossover, and no crossover (the best-performing hyper-heuristic on average). *Utility rate* indicates the percentage usage of a low-level heuristic during a run. Figure 3a shows utility rate of each heuristic considering only moves that improve on the current best-of-run solution. Figure 3b shows the average utility rate of each heuristic considering all moves (i.e., how many times each heuristic was chosen during the search process). These utility rates are average values over a single run of each instance over all 270 instances in ORLib.

In all cases there are clearly stronger low-level heuristics on average; however, this is not reflected in the amount of times each heuristic is selected overall. Because of the nature of the choice function, some low-level heuristics will be selected at a higher rate than others at certain points of the search, usually through repeated invocation. Although in percentage terms this is roughly uniform over the full benchmark dataset, it is not the case that low-level heuristic selection is uniform for a particular instance. Moreover, these figures show that all the low-level heuristics available are capable of contributing to the improvement of a solution at a given stage for at least some of the instances. This provides a justification for their continued presence in the low-level heuristic set. Similar behaviour was observed for all hyper-heuristics tested.

## 6.2 Controlling Crossover at the Domain Level for the MKP

As discussed in Section 3, the constraints of the 0-1 multidimensional knapsack problem can be relaxed to yield the related LP-relaxed version of the problem. It is known that the solutions to the LP-relaxed version of the MKP can provide good approximations for the 0-1 version of the problem (Chu and Beasley, 1998). Using the $jqdInit$ initialisation method (see Section 5) to generate a list of solutions to use as second candidate solutions for crossover, the same nine hyper-heuristics are again applied to ORLib using the same parameters as before. As before, 0.1 * $n$ solutions are generated by $jqdInit$, where $n$

Table 5: Average %-gap over all ORLib instances for each hyper-heuristic using a list of solutions to provide the second child for crossover managed at the domain level.

| | Acceptance Criteria | | |
|---|---|---|---|
| Selection Mechanism | Only Improving | Late Acceptance Strategy | Simulated Annealing |
| Simple random | 0.74 $_{0.76}$ | 0.71 $_{0.74}$ | 0.71 $_{0.76}$ |
| Choice function | 0.75 $_{0.78}$ | **0.70** $_{0.74}$ | 0.71 $_{0.76}$ |
| Reinforcement learning | 0.73 $_{0.74}$ | 0.71 $_{0.75}$ | **0.70** $_{0.76}$ |

is the number of variables in the instance currently being solved. Table 5 shows their performance in terms of %-gap over a single run of each instance of ORLib.

The best average %-gap over all ORLib instances is 0.70, obtained by choice function–late acceptance strategy and reinforcement learning–simulated annealing. An independent Student's *t*-test within a 95% confidence interval shows no statistically significant difference between these two hyper-heuristics. Interestingly, those hyper-heuristics using late acceptance strategy and simulated annealing move acceptance outperform those using only improving. This is in contrast to the hyper-heuristics discussed in Section 6.1, where crossover is controlled at the hyper-heuristic level, where only improving acceptance performed best. This is closer to what would be expected, as simulated annealing and late acceptance strategy are designed to overcome the problem of becoming trapped in local optima. Despite this, no clear conclusions can be drawn as to why this reversal of performance is observed in the case that crossover is controlled at the domain-specific level. As with the previous experiments, the acceptance criterion used has a greater effect on the quality of solutions obtained than the selection method.

Although there are two best-performing hyper-heuristics within this framework, we compare only one hyper-heuristic from each framework in the following section. We take the choice function–late acceptance strategy to compare to the best-performing hyper-heuristic (see Section 6.1) and existing methods from the literature.

## 6.3 Comparison of Hyper-heuristics Managing Crossover at the Hyper-heuristic Level and the Domain Level

Table 6 shows detailed results for each instance type for choice function–late acceptance strategy with crossover controlled at the domain-specific level and the best performing hyper-heuristic (choice function–only improving with no crossover) over the ORLib benchmarks. When comparing the performance of the two hyper-heuristics, controlling crossover at the domain-specific level results in better performance on average for 26 of the 27 sets of instances. This difference is statistically significant in 22 of these cases.

The general applicability of these hyper-heuristics is tested by applying them to two further benchmark sets, each with differing properties. SAC-94 is a set of benchmark instances from classic papers in the literature (see Section 4.5), where optimal solutions are known for each problem. It is difficult to perform a direct comparison with techniques over these instances because of the difference in termination criteria and running times. For example, some methods in the literature provide the best results over 30 runs or more. As defined by Özcan and Basaran (2009), if an algorithm finds the optimal solution in at least 5% of trial runs for a given instance, it is deemed a successful run. The success rate over each dataset is therefore the number of successful runs

Table 6: Detailed performance of choice function–late acceptance strategy with crossover managed at domain level and choice function–only improving with no crossover on ORLib instances (based on average %-gap).

| Problem Set | CF-LAS | CF-OI$_{NC}$ |
|---|---|---|
| OR5x100-0.25 | 1.16 $_{0.20}$ | 1.22 $_{0.25}$ |
| OR5x100-0.50 | 0.53 $_{0.08}$ | 0.59 $_{0.16}$ |
| OR5x100-0.75 | 0.40 $_{0.07}$ | 0.39 $_{0.08}$ |
| OR5x250-0.25 | 0.42 $_{0.04}$ | 0.51 $_{0.10}$ |
| OR5x250-0.50 | 0.20 $_{0.03}$ | 0.42 $_{0.19}$ |
| OR5x250-0.75 | 0.13 $_{0.01}$ | 0.21 $_{0.04}$ |
| OR5x500-0.25 | 0.19 $_{0.03}$ | 0.60 $_{0.13}$ |
| OR5x500-0.50 | 0.10 $_{0.03}$ | 0.85 $_{0.13}$ |
| OR5x500-0.75 | 0.06 $_{0.01}$ | 0.32 $_{0.09}$ |
| OR10x100-0.25 | 2.00 $_{0.22}$ | 2.08 $_{0.37}$ |
| OR10x100-0.50 | 1.02 $_{0.19}$ | 1.16 $_{0.15}$ |
| OR10x100-0.75 | 0.58 $_{0.08}$ | 0.66 $_{0.06}$ |
| OR10x250-0.25 | 0.83 $_{0.09}$ | 1.02 $_{0.18}$ |
| OR10x250-0.50 | 0.39 $_{0.06}$ | 0.58 $_{0.11}$ |
| OR10x250-0.75 | 0.23 $_{0.03}$ | 0.41 $_{0.06}$ |
| OR10x500-0.25 | 0.40 $_{0.06}$ | 1.10 $_{0.35}$ |
| OR10x500-0.50 | 0.18 $_{0.02}$ | 1.20 $_{0.31}$ |
| OR10x500-0.75 | 0.12 $_{0.01}$ | 0.61 $_{0.16}$ |
| OR30x100-0.25 | 3.45 $_{0.46}$ | 3.91 $_{0.57}$ |
| OR30x100-0.50 | 1.56 $_{0.26}$ | 1.85 $_{0.27}$ |
| OR30x100-0.75 | 0.92 $_{0.08}$ | 1.04 $_{0.20}$ |
| OR30x250-0.25 | 1.55 $_{0.17}$ | 2.12 $_{0.25}$ |
| OR30x250-0.50 | 0.71 $_{0.08}$ | 1.08 $_{0.14}$ |
| OR30x250-0.75 | 0.39 $_{0.04}$ | 0.52 $_{0.08}$ |
| OR30x500-0.25 | 0.92 $_{0.10}$ | 1.99 $_{0.27}$ |
| OR30x500-0.50 | 0.39 $_{0.05}$ | 1.66 $_{0.10}$ |
| OR30x500-0.75 | 0.23 $_{0.02}$ | 0.82 $_{0.15}$ |
| Average | 0.70 $_{0.09}$ | 1.07 $_{0.18}$ |

divided by the number of problems in the set. Choice function–late acceptance strategy performs a single run on each instance as before. Table 7a shows the performance of the hyper-heuristic in terms of success rate over each set of instances in SAC-94. Choice function–late acceptance strategy with crossover controlled at the domain-specific level performs at least as well as choice function–only improving with no crossover in every group of instances in this set.

The final benchmark set on which to test the hyper-heuristics is the GK set of 11 large instances provided by Glover and Kochenberger (2000). Table 7b gives the results for both hyper-heuristics as the average of 10 runs on each instance. The LP-relaxed optimal solutions are again used as a basis to derive %-gap standard deviations are included as subscripts. Choice function–only improving with no crossover performs relatively badly on this larger set of instances, obtaining an average %-gap of 0.92 compared to 0.45 obtained by the choice function–late acceptance strategy hyper-heuristic with crossover controlled at the domain-specific level.

Table 7: (a) Success rate over all SAC-94 instances, and (b) %-gap over Glover and Kochenberger instances for choice function–late acceptance strategy with domain level crossover and choice function–only improving with no crossover.

| (a) | | | | (b) | | |
|---|---|---|---|---|---|---|
| Dataset | Count | CF-LAS | CF-OI$_{NC}$ | Instance | CF-LAS | CF-OI$_{NC}$ |
| hp | 2 | 0.00 | 0.00 | GK01 | 0.57 $_{1.49}$ | 1.33 $_{6.82}$ |
| pb | 6 | 0.67 | 0.50 | GK02 | 0.81 $_{3.86}$ | 1.60 $_{9.66}$ |
| pet | 6 | 0.50 | 0.34 | GK03 | 0.63 $_{3.10}$ | 1.64 $_{18.25}$ |
| sento | 2 | 1.00 | 1.00 | GK04 | 0.91 $_{3.77}$ | 1.84 $_{18.18}$ |
| weing | 8 | 0.63 | 0.63 | GK05 | 0.45 $_{3.00}$ | 0.83 $_{13.61}$ |
| weish | 30 | 1.00 | 0.64 | GK06 | 0.76 $_{5.02}$ | 1.54 $_{23.00}$ |
| | | | | GK07 | 0.19 $_{6.48}$ | 0.33 $_{18.81}$ |
| | | | | GK08 | 0.33 $_{5.68}$ | 0.55 $_{9.57}$ |
| | | | | GK09 | 0.07 $_{7.47}$ | 0.10 $_{12.95}$ |
| | | | | GK10 | 0.14 $_{8.68}$ | 0.16 $_{14.07}$ |
| | | | | GK11 | 0.13 $_{12.34}$ | 0.15 $_{15.10}$ |
| | | | | **Average** | 0.45$_{5.54}$ | 0.92$_{14.55}$ |

Table 8: Average %-gap of (meta)heuristics and CPLEX over all instances in ORLib.

| Type | Reference | %-gap |
|---|---|---|
| MIP | CPLEX 12.5 | 0.52 |
| MA | Raidl (1998) | 0.53 |
| MA | Chu and Beasley (1998) | 0.54 |
| **Hyper-heuristic** | **CF-LAS** | **0.70** |
| MA | Özcan and Basaran (2009) | 0.92 |
| Permutation GA | Hinterding (1994); Raidl (1998) | 1.30 |
| Heuristic | Pirkul (1987) | 1.37 |
| Heuristic | Freville and Plateau (1994) | 1.91 |
| Heuristic | Qian and Ding (2007) | 2.28 |
| MIP | Chu and Beasley (1998) (CPLEX 4.0) | 3.14 |
| Heuristic | Magazine and Oguz (1984) | 7.69 |

### 6.3.1 Comparison with Previous Approaches

Table 8 shows the results of the best hyper-heuristic presented in this paper, choice function–late acceptance strategy (CF-LAS) with crossover controlled at the domain-specific level, compared to a number of techniques from the literature over the ORLib benchmarks. CPLEX (IBM, 2014) is a general-purpose mixed-integer programming (MIP) package used to solve linear optimisation problems. Chu and Beasley (1998) provided results using CPLEX 4.0 over the ORLib set of MKP benchmark instances. Here we include results for CPLEX 12.5 over ORLib, SAC-94, and the larger GK benchmarks to compare with our methods and as a benchmark for comparison for future researchers in this area. For each instance, CPLEX 12.5 is allowed to run for a maximum of 1800 CPU seconds with a maximum working memory of 8 GB.

Table 9: Performance comparison with best metaheuristic technique in the literature over ORLib instances with $n = 500$ objects.

| | Vasquez and Vimont (2005) | | **CF-LAS** | |
|---|---|---|---|---|
| Instance | %-gap | $t[s]$* | %-gap | $t[s]$ |
| OR5x500-0.25 | $0.07_{0.01}$ | 14651* | $0.19_{0.03}$ | 11 |
| OR5x500-0.50 | $0.04_{0.05}$ | 6133* | $0.10_{0.03}$ | 16 |
| OR5x500-0.75 | $0.02_{0.00}$ | 7680* | $0.06_{0.01}$ | 22 |
| OR10x500-0.25 | $0.17_{0.02}$ | 10791* | $0.40_{0.06}$ | 14 |
| OR10x500-0.50 | $0.08_{0.00}$ | 8128* | $0.18_{0.02}$ | 21 |
| OR10x500-0.75 | $0.06_{0.01}$ | 6530* | $0.12_{0.01}$ | 29 |
| OR30x500-0.25 | $0.48_{0.05}$ | 30010* | $0.92_{0.10}$ | 23 |
| OR30x500-0.50 | $0.21_{0.02}$ | 35006* | $0.39_{0.05}$ | 39 |
| OR30x500-0.75 | $0.14_{0.01}$ | 45240* | $0.23_{0.02}$ | 55 |
| **Average** | $0.14_{0.02}$ | 18241* | $\mathbf{0.29}_{0.03}$ | **26** |

From this table, it can be seen that the hyper-heuristics presented in this article perform well in comparison to many previous approaches. The use of $10^6$ fitness evaluations as a termination criterion allows direct comparison to previous metaheuristic approaches. The results for Hinterding (1994) are given as provided by Raidl (1998). The %-gap of 0.70 obtained by CF-LAS is better than the previous metaheuristic methods of Özcan and Basaran (2009) and Hinterding (1994) and a number of existing heuristic methods. The best %-gaps obtained by metaheuristics are the memetic algorithms of Chu and Beasley (1998) and the variant of their work provided by Raidl (1998).

The currently best-known results in the literature for the ORLib instances were obtained by Vasquez and Vimont (2005). Results from this study are only available for the largest instances of ORLib where $n = 500$. Results for these instances obtained using choice function–late acceptance strategy are compared with the results of Vasquez and Vimont (2005) in Table 9.

Using an independent Student's $t$-test within a 95% confidence interval, there is no statistically significant difference in performance between choice function–late acceptance strategy and the method of Vasquez and Vimont (2005) for each set of 10 instances in Table 9. A fundamental goal of hyper-heuristic research is to provide solutions that are "good enough, soon enough, cheap enough" (Burke, Hart et al., 2003). Although the work of Vasquez and Vimont (2005) was performed using inferior hardware, there is a stark contrast in execution times of each technique.[1] The results of choice function–late

[1]Note on CPU times based on Dongarra (2013):

- Intel P4 1700 MHz = 796 MFLOP/s
- Intel P4 2 GHz (estimated) 796 * 2 / 1.7 = 936.47 (scaled from 1.7 GHz to 2GHz)
- Intel Core 2 Q6600 Kensfield (1 core) 2.4 GHz = 2426 MFLOP/s
- Intel Core 2 Duo 3 GHz (estimated) 2426 * 3 / 2.4 = 3032.5 MFLOP/s (scaled from 2.4 to 3 GHz)

Based on the above Intel Core 2 Duo 3, GHz is estimated as 3032.5 / 936.47 = 3.24 times faster. $t[s]$* for Vasquez and Vimont (2005) in Table 9 are scaled using these CPU times.

Table 10: Success rate of techniques from the literature over a subset SAC-94 instances.

| Technique | Reference | sento | pet | weing |
|---|---|---|---|---|
| MIP | CPLEX 12.5 | 1.00 | 1.00 | 1.00 |
| Memetic algorithm | Chu and Beasley (1998) | 1.00 | 1.00 | 1.00 |
| Memetic algorithm | Cotta and Troya (1998) | 1.00 | 1.00 | 1.00 |
| Multimeme memetic algorithm | Özcan and Basaran (2009) | 1.00 | 0.80 | 0.50 |
| **Hyper-heuristic** | **CF-LAS** | **1.00** | **0.60** | **0.50** |
| Attribute grammar | Cleary and O’Neill (2005) | 0.50 | 0.80 | 0.50 |
| Genetic algorithm | Khuri et al. (1994) | 0.50 | 0.60 | 0.50 |
| Particle swarm optimisation | Hembecker et al. (2007) | 0.00 | — | 0.50 |
| Grammatical evolution | Cleary and O’Neill (2005) | 0.00 | 0.20 | 0.00 |

Table 11: Performance comparison of choice function–late acceptance strategy hyper-heuristic, evolutionary algorithms of Raidl and Gottlieb (2005), and CPLEX 12.5 on Glover and Kochenberger instances in terms of %-gap.

| Instance | CPLEX 12.5 | DI | **CF-LAS** | WB | PE |
|---|---|---|---|---|---|
| GK01 | 0.26 | 0.27 $_{0.03}$ | 0.57 $_{0.04}$ | 0.31 $_{0.08}$ | 0.38 $_{0.07}$ |
| GK02 | 0.45 | 0.46 $_{0.01}$ | 0.81 $_{0.10}$ | 0.48 $_{0.05}$ | 0.50 $_{0.06}$ |
| GK03 | 0.26 | 0.37 $_{0.01}$ | 0.63 $_{0.05}$ | 0.45 $_{0.04}$ | 0.52 $_{0.06}$ |
| GK04 | 0.47 | 0.53 $_{0.02}$ | 0.91 $_{0.07}$ | 0.67 $_{0.08}$ | 0.71 $_{0.09}$ |
| GK05 | 0.21 | 0.29 $_{0.00}$ | 0.45 $_{0.04}$ | 0.40 $_{0.05}$ | 0.46 $_{0.07}$ |
| GK06 | 0.32 | 0.43 $_{0.02}$ | 0.76 $_{0.07}$ | 0.61 $_{0.06}$ | 0.70 $_{0.09}$ |
| GK07 | 0.06 | 0.09 $_{0.00}$ | 0.19 $_{0.03}$ | 0.38 $_{0.08}$ | 0.52 $_{0.09}$ |
| GK08 | 0.14 | 0.17 $_{0.01}$ | 0.33 $_{0.03}$ | 0.53 $_{0.07}$ | 0.75 $_{0.09}$ |
| GK09 | 0.02 | 0.03 $_{0.00}$ | 0.07 $_{0.01}$ | 0.56 $_{0.04}$ | 0.89 $_{0.08}$ |
| GK10 | 0.04 | 0.05 $_{0.00}$ | 0.14 $_{0.02}$ | 0.73 $_{0.07}$ | 1.10 $_{0.07}$ |
| GK11 | 0.05 | 0.05 $_{0.00}$ | 0.13 $_{0.01}$ | 0.87 $_{0.06}$ | 1.24 $_{0.06}$ |
| **Average** | 0.21 | 0.25 $_{0.01}$ | **0.45** $_{0.04}$ | 0.54 $_{0.06}$ | 0.71 $_{0.08}$ |

acceptance strategy are obtained in a fraction of the time taken by Vasquez and Vimont (2005) and are less than 0.15% closer to the LP-relaxed optimum in absolute terms.

An indirect comparison between techniques can be made on a subset of the instances in SAC-94 in terms of success rate, as shown in Table 10. Three common problem instance sets from SAC-94 are used for comparison: the pet problem set, (with pet2 omitted), the sento problem set, and the last two instances of the weing problem set. The memetic algorithm of Chu and Beasley (1998) again performs well, with particle swarm optimisation and grammatical evolution performing particularly badly. Choice function–late acceptance strategy performs amicably in comparison to the results in the literature. CPLEX 12.5 finds optimal solutions for entire SAC-94 dataset using the hardware and settings outlined previously and taking a maximum of 0.3 seconds per instance.

Table 11 compares the performance of choice function–late acceptance strategy with the methods of Raidl and Gottlieb (2005) and CPLEX 12.5 using the benchmarks provided by Glover and Kochenberger (2000). Raidl and Gottlieb (2005) experimented with a number of different representations in evolutionary algorithms for the MKP. The three best results were obtained from direct representation (DI), weight-biased representation (WB), and permutation representation (PE). The results of their study are taken as averages over 30 runs and allowed to produce $10^6$ non-duplicate individuals. Standard deviations for the 30 runs of each instance by Raidl and Gottlieb are provided as subscripts. Our hyper-heuristics are also allowed $10^6$ evaluations; however, duplicate individuals are counted. The direct encoding from Raidl and Gottlieb (2005) outperforms our hyper-heuristic; however, the hyper-heuristic compares favourably to the other two encoding methods shown. Although only an indirect comparison can be made because of the differing termination criteria of each technique and subsequently their running times, CPLEX 12.5 performs particularly well on these instances, with an average %-gap of 0.21 compared to the 0.45 %-gap of the choice function–late acceptance strategy hyper-heuristic.

## 7 Conclusions

The use of crossover is still in debate in the evolutionary algorithms community. Some limited theoretical studies show that crossover is useful, and some others show they are not (Forrest and Mitchell, 1992; Jansen and Wegener, 2005). Hence, our experiments with hyper-heuristics include the control of crossover operators at the domain level and hyper-heuristic level as well as the case when it is not used. There are many alternative strategies to control crossover operators in selection hyper-heuristics. A hyper-heuristic itself is a control mechanism, since crossover operators do not have to be used at each step during the search process and this decision is made by the hyper-heuristic. In that regard, we have experimented with many algorithmic combinations of components: heuristic selection method, move acceptance criteria, and crossover operator input argument management scheme to observe the influence of different choices within selection hyper-heuristics.

Two frameworks for controlling crossover in single-point selection hyper-heuristics were presented using a common NP-hard combinatorial optimisation problem as a test bed. Crossover was included at two levels. First, it was controlled at the hyper-heuristic level without domain-specific information. Second, it was controlled below the domain barrier and given domain-specific information. In each case, a list of potential second solutions to be used in crossover was maintained. In this problem domain, crossover performs better when it is controlled below the domain barrier and problem-specific information is used. In the case where crossover control is below the domain barrier, the best hyper-heuristic tested (choice function–late acceptance strategy) is able to provide comparable performance to the state-of-the-art metaheuristics over a number of benchmark libraries. Although the management of crossover is desirable at the domain level in this case, unfortunately it is not always possible to access domain level information in other hyper-heuristic frameworks. This raises questions regarding the definition of hyper-heuristics and exactly where the responsibility of managing the arguments for low-level heuristics should lie.

When crossover is controlled at the hyper-heuristic level, dynamic acceptance criteria such as simulated annealing and late acceptance strategy are outperformed by only improving move acceptance in this domain. This difference is particularly pronounced when an intelligent scheme for managing crossover is not used. In this study the selection mechanism used does not seem to affect the quality of solutions obtained. The choice of acceptance criterion and crossover control scheme has a far greater effect on solution quality. In the case of domain-level crossover control, the performance of the acceptance criteria is reversed, with simulated annealing and late acceptance strategy outperforming only improving.

We introduced a new initialisation scheme for the MKP that allows the generation of infeasible solutions. This initialisation method was able to outperform two existing initialisation schemes as a method for providing candidate solutions for crossover within a selection hyper-heuristic on a subset of ORLib instances. As the best solutions for the MKP are known to be on the boundary between feasible and infeasible solutions, there is benefit in allowing infeasible solutions to be used as input for crossover low-level heuristics. This highlights a fundamental issue in evolutionary computation design, the ability of a fitness function to accurately reflect the quality of a solution with respect to some unknown optimum. Results using CPLEX 12.5 were included over the three benchmark libraries for the use of future researchers in this area. Although the generality of the hyper-heuristics in this paper was demonstrated using different benchmarks, it would be interesting to analyse the performance of these frameworks over a number different problem domains. Generality does not necessarily need to be shown over the problem domains used. It is possible to classify low-level heuristics with different characteristics, that is, mutation heuristics, and to group multiple low-level heuristics into sets. The performance of hyper-heuristics using different sets of low-level heuristics, representing different possible experimental conditions, can demonstrate a different flavour of generality.

## References

Angelelli, E., Mansini, R., and Grazia Speranza, M. (2010). Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers & Operations Research*, 37(11): 2017–2026.

Bai, R., Blazewicz, J., Burke, E. K., Kendall, G., and McCollum, B. (2012). A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR: A Quarterly Journal of Operations Research*, 10(1): 43–66.

Battiti, R., Brunato, M., and Mascia, F. (Eds.) (2008). *Reactive search and intelligent optimization*. New York: Springer.

Bilgin, B., Özcan, E., and Korkmaz, E. E. (2006). An experimental study on hyper-heuristics and exam timetabling. In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, pp. 394–412. Lecture Notes in Computer Science, Vol. 3867.

Boussier, S., Vasquez, M., Vimont, Y., Hanafi, S., and Michelon, P. (2010). A multi-level search strategy for the 0-1 multidimensional knapsack problem. *Discrete Applied Mathematics*, 158(2): 97–109.

Burke, E. K., and Bykov, Y. (2008). A late acceptance strategy in hill-climbing for exam timetabling problems. In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)*.

Burke, E. K., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., and Vazquez-Rodriguez, J. A. (2009). HyFlex: A flexible framework for the design and analysis of hyper-heuristics. In *Proceedings of the Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2009)*, pp. 790–797.

Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P., and Schulenburg, S. (2003). Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, (Eds.), *Handbook of metaheuristics*, pp. 457–474. New York: Kluwer.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64:1695–1724.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In C. L. Mumford and L. C. Jain (Eds.), *Computational intelligence: Collaboration, fusion and emeregense*, pp. 177–201. New York: Springer.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In M. Gendreau and J.-Y. Potvin (Eds.), *Handbook of metaheuristics*, 2nd ed., pp. 449–468. New York: Springer.

Burke, E. K., Kendall, G., Misir, M., and Özcan, E. (2012). Monte Carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196(1): 73–90.

Burke, E. K., Kendall, G., and Soubeiga, E. (2003). A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6): 451–470.

Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. (2007). A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176(1): 177–192.

Chakhlevitch, K., and Cowling, P. (2005). Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, pp. 23–33. *Lecture Notes in Computer Science*, Vol. 3448.

Chakhlevitch, K., and Cowling, P. (2008). Hyperheuristics: Recent developments. In C. Cotta, M. Sevaux, and K. Sörensen (Eds.), *Adaptive and multilevel metaheuristics*, pp. 3–29. New York: Springer.

Chu, P. C., and Beasley, J. E. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1): 63–86.

Cleary, R., and O'Neill, M. (2005). An attribute grammar decoder for the 0/1 multiconstrained knapsack problem. In *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, pp. 34–45. *Lecture Notes in Computer Science*, Vol. 3448.

Cobos, C., Mendoza, M., and Leon, E. (2011). A hyper-heuristic approach to design and tuning heuristic methods for web document clustering. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2011)*, pp. 1350–1358.

Cotta, C., and Troya, J. (1998). A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In *Artificial Neural Nets and Genetic Algorithms: Proceedings*, pp. 250–254.

Cowling, P., Kendall, G., and Soubeiga, E. (2001a). A hyperheuristic approach to scheduling a sales summit. In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000)*, pp. 176–190. *Lecture Notes in Computer Science*, Vol. 2079.

Cowling, P., Kendall, G., and Soubeiga, E. (2001b). A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the Metaheuristics International Conference (MIC 2001)*, pp. 127–131.

Croce, F. D., and Grosso, A. (2012). Improved core problem based heuristics for the 0-1 multi-dimensional knapsack problem. *Computers & Operations Research*, 39(1): 27–31.

Demeester, P., Bilgin, B., Causmaecker, P. D., and Berghe, G. V. (2012). A hyperheuristic approach to examination timetabling problems: Benchmarks and a new problem from practice. *Journal of Scheduling*, 15(1): 83–103.

Dongarra, J. J. (2013). Performance of various computers using standard linear equations software. Retrieved from http://www.netlib.org/benchmark/performance.pdf

Drake, J. H., Kililis, N., and Özcan, E. (2013). Generation of VNS components with grammatical evolution for vehicle routing. In *Proceedings of Genetic Programming: 16th European Conference (EuroGP 2013)*, pp. 25–36. Lecture Notes in Computer Science, Vol. 7831.

Drake, J. H., Özcan, E., and Burke, E. K. (2012). An improved choice function heuristic selection for cross domain heuristic search. In *Proceedings of Parallel Problem Solving from Nature (PPSN 2012)*, pp. 307–316.

Fialho, Á., Costa, L. D., Schoenauer, M., and Sebag, M. (2008). Extreme value based adaptive operator selection. In *Proceedings of Parallel Problem Solving from Nature (PPSN 2008)*, pp. 175–184. Lecture Notes in Computer Science, Vol. 5199.

Fisher, M., and Thompson, G. (1961). Probabilistic learning combinations of local jobshop scheduling rules. In *Proceedings of the Factory Scheduling Conference*, Carnegie Institute of Technology.

Forrest, S., and Mitchell, M. (1992). Relative building block fitness and the building block hypothesis. In *Proceedings of Foundations of Genetic Algorithms (FOGA 1992)*, pp. 109–126.

Freville, A., and Plateau, G. (1994). An efficient preprocessing procedure for the multidimensional 0-1 knapsack problem. *Discrete Applied Mathematics*, 49(1-3): 189–212.

García-Villoria, A., Salhi, S., Corominas, A., and Pastor, R. (2011). Hyper-heuristic approaches for the response time variability problem. *European Journal of Operational Research*, 211(1): 160–169.

Garey, M. R., and Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. New York: W. H. Freeman.

Glover, F., and Kochenberger, G. (2000). Benchmarks for "the multiple knapsack problem." Retrieved from http://hces.bus.olemiss.edu/tools.htm

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Boston: Addison-Wesley.

Gottlieb, J. (2000). On the effectivity of evolutionary algorithms for the multidimensional knapsack problem. In *Proceedings of Artificial Evolution (AE 1999)*, pp. 23–37. Lecture Notes in Computer Science, Vol. 1829.

Hanafi, S., Lazic, J., Mladenovic, N., Wilbaut, C., and Crevits, I. (2010). New hybrid matheuristics for solving the multidimensional knapsack problem. In *Proceedings of the International Conference on Hybrid Metaheuristics (HM 2010)*, pp. 118–132. Lecture Notes in Computer Science, Vol. 6373.

Hanafi, S., and Wilbaut, C. (2011). Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Annals of Operations Research*, 183(1): 125–142.

Hembecker, F., Lopes, H. S., and Godoy, Jr., W. (2007). Particle swarm optimization for the multidimensional knapsack problem. In *Proceedings of the International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2007)*, pp. 358–365. Lecture Notes in Computer Science, Vol. 4431.

Hinterding, R. (1994). Mapping, order-independent genes and the knapsack problem. In *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC 1994)*, pp. 13–17.

Huberman, B. A., Lukose, R. M., and Hogg, T. (1997). An economics approach to hard computational problems. *Science*, 275(5296): 51–54.

Hyde, M. (2010). *A genetic programming hyper-heuristic approach to automated packing*. Unpublished doctoral dissertation, University of Nottingham, UK.

IBM (2014). CPLEX optimizer. Retrieved from http://www.ibm.com/software/commerce/optimization/cplex-optimizer/

Jackson, W. G., Özcan, E., and Drake, J. H. (2013). Late acceptance-based selection hyper-heuristics for cross-domain heuristic search. In *Proceedings of the 13th Annual Workshop on Computational Intelligence (UKCI 2013)*, pp. 228–235.

Jansen, T., and Wegener, I. (2005). Real royal road functions: Where crossover provably is essential. *Discrete Applied Mathematics*, 149(1-3): 111–125.

Jones, T. (1995) Crossover, macromutation, and population-based search. In *Proceedings of the International Conference on Genetic Algorithms (ICGA 1995)*, pp. 73–80.

Kheiri, A., and Özcan, E. (2013). A hyper-heuristic with a round robin neighbourhood selection. In *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2013)*, pp. 1–12. Lecture Notes in Computer Science, Vol. 7832.

Khuri, S., Bäck, T., and Heitkötter, J. (1994). The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the ACM Symposium on Applied Computing (SAC '94)*, pp. 188–193.

Kiraz, B., Uyar, A. S., and Özcan, E. (2013). Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*, 64(1): 1753–1769.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598): 671–680.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Magazine, M. J., and Oguz, O. (1984). A heuristic algorithm for the multidimensional zero-one knapsack problem. *European Journal of Operational Research*, 16(3): 319–326.

Maniezzo, V., Stützle, T., and Voss, S. (Eds.) (2010). *Matheuristics: Hybridizing metaheuristics and mathematical programming*. New York: Springer.

Mansini, R., and Speranza, M. G. (2012). CORAL: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, 24(3): 399–415.

Maturana, J., Lardeux, F., and Saubion, F. (2010). Autonomous operator management for evolutionary algorithms. *Journal of Heuristics*, 16(6): 881–909.

Misir, M., Verbeeck, K., Causmaecker, P. D., and Berghe, G. V. (2012). An intelligent hyper-heuristic framework for CheSc 2011. In *Proceedings of Learning and Intelligent Optimization (LION 2012)*, pp. 461–466. Lecture Notes in Computer Science, Vol. 7219.

Moscato, P., Cotta, C., and Mendes, A. (2004). Memetic algorithms. In G. C. Onwubolu and B. V. Babu (Eds.), *New optimization techniques in engineering*. New York: Springer.

Nareyek, A. (2001). Choosing search heuristics by non-stationary reinforcement learning. In M. Resende and J. Pinho de Sousa (Eds.), *Metaheuristics: Computer decision-making*, pp. 523–544. New York: Kluwer.

Nenad, M., and Pierre, H. (1997). Variable neighborhood search. *Computers and Operations Research*, 24(11): 1097–1100.

Ochoa, G., and Hyde, M. (2011). The cross-domain heuristic search challenge (CHeSC 2011). Retrieved from http://www.asap.cs.nott.ac.uk/chesc2011/

Ong, Y.-S., Lim, M.-H., Zhu, N., and Wong, K.-W. (2006). Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on Systems, Man and Cybernetics Part B*, 36(1): 141–152.

Onsem, W. V., and Demoen, B. (2013). ParHyFlex : A framework for parallel hyper-heuristics. In *Proceedings of the 25th Benelux Artificial Intelligence Conference*, pp. 231–238.

Özcan, E., and Basaran, C. (2009). A case study of memetic algorithms for constraint optimization. *Soft Computing*, 13(8-9): 871–882.

Özcan, E., Bilgin, B., and Korkmaz, E. E. (2006). Hill climbers and mutational heuristics in hyperheuristics. In *Proceedings of Parallel Problem Solving from Nature (PPSN 2006)*, pp. 202–211. Lecture Notes in Computer Science, Vol. 4193.

Özcan, E., Bilgin, B., and Korkmaz, E. E. (2008). A comprehensive analysis of hyper heuristics. *Intelligent Data Analysis*, 12(1): 3–23.

Özcan, E., Bykov, Y., Birben, M., and Burke, E. K. (2009). Examination timetabling using late acceptance hyper-heuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 997–1004.

Pirkul, H. (1987). A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics*, 34(2): 161–172.

Puchinger, J., Raidl, G. R., and Pferschy, U. (2006). The core concept for the multidimensional knapsack problem. In *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006)*, pp. 195–208. Lecture Notes in Computer Science, Vol. 3906.

Qian, F., and Ding, R. (2007). Simulated annealing for the 0/1 multidimensional knapsack problem. *Numerical Mathematics*, 16(4): 320–327.

Raidl, G. R. (1998). An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Proceedings of the IEEE Conference on Evolutionary Computation (CEC 1998)*, pp. 207–211.

Raidl, G. R., and Gottlieb, J. R. (2005). Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4): 441–475.

Raidl, G. R., and Puchinger, J. (2008). Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In C. Blum, M. Aguilera, A. Roli, and M. Sampels (Eds.), *Hybrid metaheuristics*, pp. 31–62. Studies in Computational Intelligence, Vol. 114. Berlin: Springer.

Rendl, A., Prandtstetter, M., Hiermann, G., Puchinger, J., and Raidl, G. R. (2012). Hybrid heuristics for multimodal homecare scheduling. In *Proceedings of Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2012)*, pp. 339–355. Lecture Notes in Computer Science, Vol. 7298.

Ross, P. (2005). Hyper-heuristics. In E. K. Burke and G. Kendall (Eds.), *Search methodologies: Introductory tutorials in optimization and decision support technologies*, pp. 529–556. New York: Springer.

Sutton, R., and Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Swan, J., Özcan, E., and Kendall, G. (2011). Hyperion: A recursive hyper-heuristic framework. In *Proceedings of Learning and Intelligent Optimization (LION 5)*, pp. 616–630. Lecture Notes in Computer Science, Vol. 6683.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 2–9.

Tavares, J., Pereira, F., and Costa, E. (2008). Multidimensional knapsack problem: A fitness landscape analysis. *IEEE Transactions on Systems, Man and Cybernetics Part B*, 38(3): 604–616.

Vasquez, M., and Hao, J. (2001). A hybrid approach for the 0-1 multidimensional knapsack problem. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pp. 328–333.

Vasquez, M., and Vimont, Y. (2005). Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1): 70–81.

Vimont, Y., Boussier, S., and Vasquez, M. (2008). Reduced costs propagation in an efficient implicit enumeration for the 0-1 multidimensional knapsack problem. *Journal of Combinatorial Optimisation*, 15(2): 165–178.

Weingartner, H. M., and Ness, D. N. (1967). Methods for the solution of the multidimensional 0/1 knapsack problem. *Operations Research*, 15(1): 83–103.