A Case Study of Controlling Crossover in a Selection Hyper-heuristic Framework with MKP

John H. Drake psxjd2@nottingham.ac.uk
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

Abstract

In evolutionary algorithms, crossover operators are used to recombine multiple candidate solutions to yield a new solution that hopefully inherits good genetic material. Hyper-heuristics are high-level methodologies which operate on a search space of heuristics for solving complex problems. 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. Crossover is increasingly being included in general purpose hyper-heuristic tools such as HyFlex and Hyperion, however little work has been done 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. Firstly, crossover is controlled at the hyper-heuristic level where no problem-specific information is required. Secondly, it is controlled at the problem domain level where problem-specific information is used to produce good quality solutions to use for 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 which aim to solve computationally difficult problems. Unlike traditional techniques, a hyper-heuristic operates on a search space of low-level heuristics rather than directly on the search space of solutions. The term 'hyper-heuristic' was first used by Denzinger et al. (1996) to describe a technique choosing and combining a number of artificial intelligence methods. Although

the term hyper-heuristic was first used at this time, ideas exhibiting hyper-heuristic behaviour can be traced back as early as 1961 (Fisher and Thompson, 1961). Hyper-heuristics have been applied successfully to a wide range of problems such as examination timetabling (Özcan et al., 2009, 2010; Burke et al., 2003b), production scheduling (Fisher and Thompson, 1961), nurse scheduling (Burke et al., 2003b) and vehicle routing (Garrido and Castro, 2009). A recent definition of hyper-heuristics is offered by Burke et al. (2010a,b):

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

This terminology includes systems which use high level strategies other than heuristics within the definition of hyper-heuristics. It also covers the two main classes of hyper-heuristics, those concerned with heuristic *selection* and those with heuristic *generation*. Here we will be working 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 criteria is met.

Crossover is often used in population-based metaheuristics as a mechanism to recombine multiple solutions. Two or more candidate solutions are selected from a population and a new solution is generated containing material from all parents. The intention is that only the best quality solutions will be kept in the population containing a mixture of material from previous solutions. This causes a problem in single-point search as each operator requires two parents as input. A trivial choice for one parent in a single-point selection hyper-heuristic is the current solution however some choice must be made when considering the second candidate solution. Many modern hyper-heuristic frameworks such as HyFlex (Burke et al., 2009a) and Hyperion (Swan et al., 2011) now include crossover operators as part of the set of low-level heuristics. Limited work has been done to research methods to select the solutions to use as input for such operators.

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 which 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 benchmark to compare the proposed frameworks for crossover control.

Section 2 provides an overview of hyper-heuristics, a definition and brief classification of hyper-heuristics 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 testbed. Section 5 details the parameter tuning necessary before the full experimentation. 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 outlined in Burke et al. (2010b). The first category contains those methodologies which select low-level heuristics to apply from a set of existing heuristics. The second one contains methodologies which create new heuristics from a set of components of other existing low-level heuristics (Burke et al., 2009b). These categories are then further broken down to distinguish between hyper-heuristics which construct solutions (Burke et al., 2007) and those which aim to improve complete solutions through local search (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. (2010a) 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 Neighbourhood 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 each of these techniques for the interested reader.

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 will be labelled $selection\ method$ - $acceptance\ criteria$ hereafter in this paper with acronyms used where space is restricted. The 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.

Cowling et al. (2001a) experimented with a number of heuristic selection mechanisms including Simple Random and Choice Function using two simple move acceptance criteria, accept All Moves and accept Only Improving moves. In the initial work, these parameters were set values however these can be changed adaptively using the method described by Cowling et al. (2001b). In this early work, the Choice Function selection combined with All Moves acceptance was shown to work well. The choice function has been shown as a successful selection mechanism in a number of further studies (Bilgin et al., 2006; Blazewicz et al., 2013; Kiraz et al., 2013).

Nareyek (2001) analysed a number of weight adaptation functions and two simple selection methods when using Reinforcement Learning (Sutton and Barto, 1998) as a selection mechanism. Taking the low-level heuristic with the maximum utility value rather than using a weighted probability of selection and using a simple additive and subtractive weight adaptation were shown to be reasonable choices when using Reinforcement Learning as a heuristic selection method.

Özcan et al. (2009) used Late Acceptance Strategy hill climbing within a single-point search hyper-heuristic framework to solve standard benchmarks of the examination timetabling problem. This work suggested that Late Acceptance Strategy was

relatively successful when used with Simple Random selection and less suitable when used with more intelligent methods such as Choice Function and Reinforcement Learning.

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 constructive hyper-heuristics for this problem, a set of single-point search hyper-heuristics were tested. Simple Random, Greedy and two Probability-based 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 superior than 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 selection methods 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 using Simple Random selection with the same move acceptance criteria. In other words, an 'intelligent' mechanism is unable to learn which heuristic to apply at a given time suggests a complex relationship between selection method and acceptance criteria that merits further investigation.

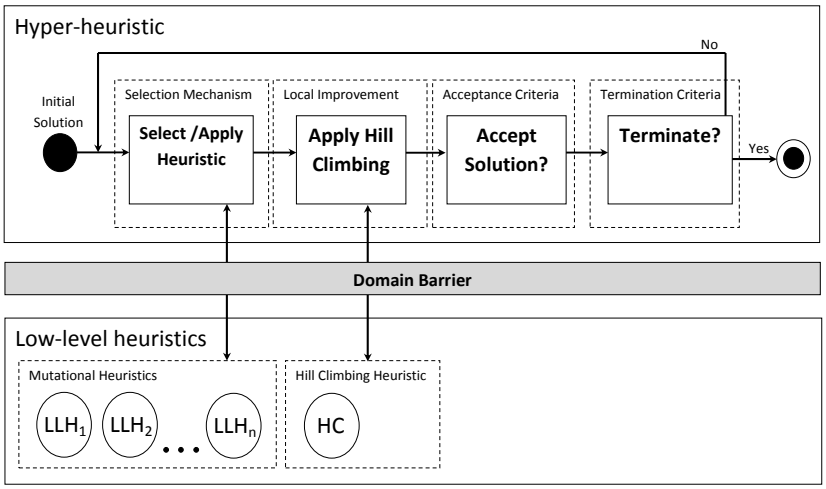
Demeester et al. (2012) used Simple Random hyper-heuristics to solve three exam timetabling datasets. Improving or Equal, Great Deluge, Simulated Annealing, Late Acceptance Strategy and Steepest Descent Late Acceptance Strategy were used as move acceptance criteria. The Simple Random - Simulated Annealing hyper-heuristic improved on a number of best results from the literature over the Toronto benchmark dataset and performed well over a second dataset provided by the authors. Other hyper-heuristics using Simulated Annealing as an acceptance criterion have been applied to a number of domains including the multimodal homecare scheduling problem (Rendl et al., 2012), DNA sequence prediction (Blazewicz et al., 2013), bin packing 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 selection mechanisms and move acceptance criteria exist in the literature. As a complete description of all selection mechanism and move acceptance criteria is beyond the scope of this paper, a number of survey papers (Burke et al., 2010a; 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. F_A is the traditional hyper-heuristic framework where a low-level heuristic is selected and applied and subsequently accepted or rejected based the quality of the move. 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 is then applied before a decision whether to accept or reject the move is made. F_C selects and applies a mutational heuristic $LLH_i \in LLH_1, ..., LLH_n$, where n is the number of mutational heuristics available, followed by a pre-defined hill climber HC before deciding whether to accept the new solution. Such a framework is illustrated in Figure 1 and is the framework we will use in this paper. F_D distinctly separates mutational heuristics and hill climbers into

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



two groups. A mutational heuristic is chosen and applied from the first group and accepted or rejected based on performance. A hill climber is then chosen from the second group 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 mechanism. Memetic Algorithms have previously shown to be successful on a number of different problem domains including the MKP (Chu and Beasley, 1998; Özcan and Basaran, 2009). In this paper we will analyse the effect of introducing crossover into an F_C hyper-heuristic framework.

2.3 Crossover in single-point search hyper-heuristics

Evolutionary algorithms (EAs) are a class of techniques commonly used to solve optimisation problems. An EA maintains a population of individuals which are recombined and/or mutated in order to form the next generation of the population. The intention of this process is to gradually improve the quality of the individuals in the population. The quality of a solution is measured by a *fitness function* also known as an evaluation or cost function. The associated *fitness* value of an individual is a measure to distinguish good solutions from bad solutions. The improvement in the population is achieved through the processes of selection, recombination (crossover) and mutation. Given two suitably fit individuals (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 both parents. There are a large number of crossover operators proposed in the literature for general and specific purposes.

Despite the introduction of crossover operators into modern hyper-heuristic frameworks such as HyFlex (Burke et al., 2009a) and Hyperion (Swan et al., 2011), limited research effort has been directed at managing this type of low-level heuristic. In a recent competition (Ochoa and Hyde, 2011) based on the HyFlex framework, only

two of the top ten entrants provided a description of a strategy to control arguments for crossover. Of these two, one simply uses the best seen solution so far as the second argument and the other, the eventual competition winner (Misir et al., 2012), provides a detailed explanation of a crossover management scheme. This hyper-heuristic maintains a memory of the 5 best solutions seen so far of which a random solution is used each time a crossover low-level heuristic is chosen. When a new best-of-run solution is found it replaces one of the 5 solutions in memory chosen at random. More recently Kheiri and Özcan (2013) used a simple scheme to manage the second arguments for low-level heuristics, again using the HyFlex framework. A circular queue containing the best solutions seen so far is maintained to provide second arguments for crossover operators however the length of the queue is arbitrarily set. A pointer indicates which solution is to be used each time a crossover heuristic requires a second argument and is advanced to the next solution in the queue after each application of crossover.

The methods discussed above relate to hyper-heuristics which manage the arguments for crossover operators 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 the single-point search framework used in (Kheiri and Özcan, 2013; Misir et al., 2012), 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 arguments 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 operator to use at each step in evolutionary algorithms for SAT. Although the choice of operator is made at the hyper-heuristic level, the selection of arguments for the crossover operator selected is performed at the domain level. Using two-parent crossover for all of the operators available, the individuals are selected using two schemes. In the early experimentation, this selection is performed randomly between all individuals in the population. A fitness-biased selection scheme is also used however the details of this mechanism are not explained.

The management of the second input argument required for crossover is not considered an important part of hyper-heuristic design by many researchers; indeed there are no standard mechanisms defined for controlling crossover in this context. An open research question is whether the responsibility of providing arguments for crossover and other multi-argument operators in selection hyper-heuristics should be the responsibility of the high-level hyper-heuristic or the low-level heuristics operating below the domain barrier. This is 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 (MKP) (Weingartner and Ness, 1967) is a generalised case of the 0-1 knapsack problem whose roots can be traced back to capital budgeting (Lorie and Savage, 1955) and project selection (Petersen, 1967) applications. The MKP is a resource allocation model whose objective is to select a subset of objects which yield the greatest profit whilst observing the constraints on knapsack capacities. Each object n carries a different weight in each knapsack and when selected, consumes resources in each dimension m.

Formally the MKP can be stated as:

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

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

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

where p_j is the profit for selecting item j, a_{ij} is the resource consumption of item j in dimension i, b_i is the capacity constraint of each dimension i. Using direct binary encoding, x_1,...,x_n is a set of decision variables indicating whether or not object j is included in the knapsack. 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 and single constraint equivalent. 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), Kernal 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 successfully 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; Fleszar and Hindi, 2009; 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:

$$0 \leq x_j \leq 1, \qquad j = 1, ..., n \tag{4}$$

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 based on the dual variables of the LP-relaxed problem. Raidl (1998) used a similar method which used the actual values of the LP-relaxed solution when repairing candidate solutions. Puchinger et al. (2006) explore the core concept for the MKP. The core concept reduces the problem to a subset of decision variables which are the most difficult to decide whether or not 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 Neighborhood Search showed better results than when applied to the original problem directly. Vasquez and Vimont (2005) provide the best

known results for the largest instances from the benchmarks of Chu and Beasley (1998). This approach applies Tabu Search to promising areas of the search space derived from LP-relaxed optima with the improved algorithm fixing additional variables matching the attributes of a 'good' solution.

4 Controlling crossover in selection hyper-heuristics for the Multidimensional Knapsack Problem

As interest in hyper-heuristic research increases, the use of general purpose hyper-heuristic frameworks such as HyFlex (Burke et al., 2009a) and Hyperion (Swan et al., 2011) is growing. Such frameworks often contain crossover low-level heuristics however the management of these operators it often overlooked, despite being a cornerstone of evolutionary computation for a considerable period of time. The majority of the top ten entrants to CHeSC2011 do not provide methods for managing crossover arguments although such operators are available to these hyper-heuristics. As the definition of hyper-heuristics creates a distinct separation between the high-level search strategy and the problem domain level, the question of which level should be responsible for managing the arguments required for crossover operators is an open research issue. The multidimensional knapsack problem (MKP) has been chosen as a testbed for two reasons. Firstly, as the MKP can be represented as a binary bitstring, a large number of general low-level heuristics already exist in the literature. Secondly, a large number of different benchmark datasets exist for this problem. The availability of these benchmarks allows us to test the generality of the methods we use over a wide variety of problem instances within a single problem domain.

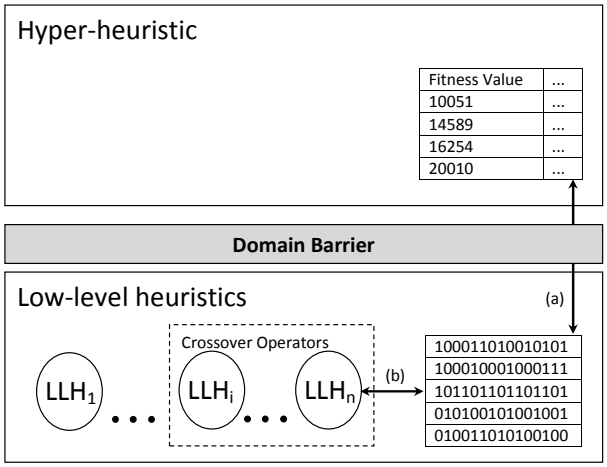
4.1 Controlling crossover in selection hyper-heuristics

Traditionally crossover is included in population-based approaches, as opposed to the single-point approaches 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. This causes a problem when considering where the second candidate solution in single-point search as each operator requires two parents as input. As it is not obvious where the second candidate solution for crossover should be managed in a single-point selection hyper-heuristic, 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 $LLH_i, ..., LLH_n$ 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 decides which solution to use for crossover based on feedback given during the search. This raises a number of questions including: which information should be passed back to the hyper-heuristic, how should this list be maintained and 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.

Figure 2: A general framework for controlling crossover with hyper-heuristic control shown by arrow (a) and low-level control shown by arrow (b)



4.1.1 Controlling crossover at the hyper-heuristic level

Candidate solutions for use as input arguments for crossover operators can be controlled at the hyper-heuristic level. In *Memory Crossover*, a list of solutions which were the best-of-run when found is maintained, from which second parents are chosen for crossover operators. 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 successfully 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 and the highest quality solution in a pair is kept and the other 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 parents to be the current solution in the hyper-heuristic and a randomly generated bitstring. This does not fit in with the original ethos of crossover which is to preserve and exploit the good characteristics of suitably fit parents. In this case, the list of potential solutions for crossover is a single random solution. In addition, each hyper-heuristic is also tested with crossover low-level heuristics omitted completely

4.1.2 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 use 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 parent 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) where 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 the optimal solutions which are not considered to be included.

Here 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 a solution with no items selected, each item j is considered sequentially from left to right. An item is included in the solution with probability equal to the items value in the LP-relaxed solution irrespective of whether a feasible solution is obtained or not. Pseudo-random 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 Algorithm to generate 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

4.2 Hyper-heuristic components

The selection methods and acceptance criteria used in this paper are introduced in Section 4.2.1 and Section 4.2.2 respectively.

4.2.1 Selection mechanisms

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

The **Choice Function (CF)** is a more elegant selection method which scores heuristics based on a combination of three different measures and applies 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 $h_1, h_2, ..., h_j$ 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, $T_n(h_j)$ is the time spent calling the heuristic for each previous invocation n of heuristic h_j and α 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 pair-wise 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 2:

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

where $I_n(h_k, h_j)$ is the change in evaluation function, $T_n(h_k, h_j)$ is the time spent calling the heuristic for each previous invocation n of heuristic h_j following h_k and β 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.

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

As mentioned previously, a score for each heuristic is given in order to rank heuristics. This score is 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 α and β weight f_1 and f_2 respectively to provide sufficient intensification of the search process and δ weights f_3 to provide sufficient diversification.

Reinforcement Learning (RL) 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. Conversely, 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.

4.2.2 Move acceptance criteria

Only Improving (OI) is a simple move acceptance criterion which accepts any improving move made by application of a low-level heuristic chosen by the selection method.

Simulated Annealing (SA) (Kirkpatrick et al., 1983) is a generic metaheuristic technique for optimisation often used as an acceptance criterion in hyper-heuristics. 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 poorer solution, the move is accepted probabilistically based on the decrease in solution quality and the decreasing temperature over time. The acceptable level of worsening solutions will decrease as the temperature decreases and the probability of moving to a worse solution will reduce over time. This probability p is given as:

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

where Δ is the change in fitness function value and T is the current temperature value.

Late Acceptance Strategy (LAS) Burke and Bykov (2008) promotes a general trend of improvement throughout a search process by comparing a candidate solution to one

generated a specified number of steps before kept in memory. 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 previous accepted solution replaces the old solution.

4.3 Low-level heuristics

A set of standard low-level heuristics from the literature have been implemented for our hyper-heuristics to operate on. In the case of a crossover operator, 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 criteria. The low-level heuristics used are as follows:

One-point Crossover (1PX) (Goldberg, 1989), given two candidate solutions, selects a single crossover point at random and exchanges the genetic data that appears on one side of this point between the two individuals.

Two-point Crossover (2PX) (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.

Uniform Crossover (UX) (Syswerda, 1989) considers each position within two chosen candidate solutions and exchanges each bit with a given exchange probability p_e, set at 0.5.

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

Parameterised Mutation (PARAxx) inverts a specified number of bits within a string. 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.

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 either 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) adopted 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. Using these relevance values a simple 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 and the relevance weights of Chu and Beasley (1998) based on dual variable values is observed to be the best efficiency measure for the MKP. 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.

4.4 Experimental data and test framework definitions

The MKP has been chosen as a suitable test domain as a number of different benchmark libraries are available with varying properties. This means different types of problems can be tested without the need to change the problem domain. SAC-94 is a standard benchmark library 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 and n ranging from 10 to 105 with optimal solutions known for all. Chu and Beasley (1998) noted that the SAC-94 instances are too small to draw meaningful conclusions of an algorithms performance from, leading to the proposal of the ORLib instances. This is 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 (nd) including much larger instances with n between 100 and 2500 and m between 15 and 100. Again no optimal solutions are known so performance is measured in terms of %-gap. All instances are available in a unified format from `http://www.cs.nott.ac.uk/~jqd/mkp/index.html`.

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 as defined in Section 4.2.2. 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. All hyper-heuristic experiments were carried out on an Intel Core 2 Duo 3 GHz CPU with 2 GB memory.

4.5 Fitness Function

As discussed previously 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

Table 1: Average %-gap over ORLib subset for each memory size

	Memory Size		
Instance Size (n)	0.10*n	0.25*n	0.50*n
OR30x100-0.75	1.03	1.05	1.10
OR30x250-0.75	0.57	0.56	0.59
OR30x500-0.75	0.77	0.73	0.77
Average	0.79	0.78	0.82

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.

5 Parameter tuning

5.1 Finding a suitable memory size for Memory Crossover

Preliminary experiments are carried out in order to ascertain an appropriate size for the memory to be used in the Memory Crossover framework. As the ORLib benchmark set contains instances with $n \in \{100, 250, 500\}$ variables, the best memory size may be different for different instance sizes. A subset of 30 ORLib instances where m = 30 and the $tightness\ ratio$ = 0.75 for each object $n \in \{100, 250, 500\}$ is used. A single run of a Simple Random - Only Improving hyper-heuristic is performed on each set of instances with each memory size and the performance in terms of %-gap reported. Three different memory lengths are tested, with $length = \{0.1 * n, 0.25 * n, 0.5 * n\}$. In all cases tournament selection with tournament size = 2 is used to select which solution to choose from the memory when a second solution is required for crossover. The complete set of 7 low-level heuristics are available to the hyper-heuristic. The results in terms of average %-gap are presented in Table 1.

These results do not show a clear distinction between using different memory lengths although the two smaller memory sizes perform slightly better than $0.50 * n$. There is also no clear correlation between the size of memory used and the size of the instance being solved. In order to assess statistical significance in this paper an independent student's t-test within a 95% confidence interval is performed for a given set of instances. In this case there is no statistically significant difference in performance between each of the three memory lengths. In light of this, memory size $0.1 * n$ is used for the following experiments. This will ensure poor quality solutions which are found early in the search are removed from the list quickly in favour of better quality solutions.

5.2 Finding a suitable initialisation method for the list of solutions

The three initialisation techniques described in Section 4.1.2 (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 Simple Random - Only Improving. Again the hyper-heuristic is allowed to run for 10^6 fitness evaluations on each instance. Table 2 details the average of the best solution in the list for each initialisation method and Table 3 shows the average solution quality of all solutions in the list over each set of 10 instances. Standard deviations are given as subscripts. The average best solution and average list quality when using R* is far superior to C*. This is unsurprising as R* was designed to generate solutions which are closer to the optimal than those generated with C*. Again an independent student's

t-test within a 95% confidence interval is performed to assess statistical significance. The best solutions produced by $jqdInit$ are also superior to C* on average with this difference being statistically significant in all cases except for OR5x100-0.25, for some instances in this dataset $jqdInit$ would not produce any feasible solutions. The best solutions produced by $jqdInit$ only slightly poorer on average than those produced by R*, this difference is 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 are statistically significantly worse quality than both C* and R*. As these solutions may still contain the 'building blocks' of good quality solution this does not affect the final solution quality as seen in Table 4. This suggests that infeasible solutions can help the search process and highlights the importance of including infeasible solutions when solving the MKP as optimal solutions are known to be close to the boundary of feasibility.

Table 2: 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 $_{46.57}$
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 3: 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 $_{62.05}$

Table 4 shows the results in terms of %-gap as the average over 10 instances for each instance set with standard deviations included as subscripts. Interestingly on these larger 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

Table 4: 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 $_{0.08}$

0.43. Despite both the average and best solutions produced by the R* initialisation being better than the $jqdInit$ in all of the datasets tested, the new initialisation method leads to better results overall after a full hyper-heuristic run.

The key difference between the previous methods and the proposed initialisation 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 as seen in Table 4. 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. As $jqdInit$ 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 both the hyper-heuristic level and the domain level. In each case, the hyper-heuristics are initially tuned and 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 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 Random Crossover, with Memory Crossover and No Crossover. Table 5 shows the performance of each hyper-heuristic over all ORLib instances using each of the crossover management strategies with standard deviations included as subscript. In this table the acronyms introduced in Section 4.2 are used for each $selection\ method$-$acceptance\ criteria$ combination.

The best performing hyper-heuristic was 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 statistically significant difference between the performance of the 27 hyper-heuristics. Using 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

Table 5: 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 $_{1.10}$

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 acceptance, all three crossover types perform similarly with no statistically significant difference between results. When using Late Acceptance Strategy and Simulated Annealing as 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 significantly from the hyper-heuristics using Only Improving 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 criteria there is little difference in the results obtained by using a different selection mechanism. However, there is significant difference between the results obtained using different acceptance criteria. This suggests that the acceptance criteria used has a more significant impact on the performance of a hyper-heuristic than selection mechanism using this heuristic set. This behaviour was also 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 3(a) shows utility rate of each heuristic considering only moves which improve on the current best-of-run solution. Figure 3(b) 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 se-

Figure 3: Average low-level heuristic utilisation for Choice Function - Only Improving hyper-heuristics with Random, Memory and No Crossover over all instances in ORLib

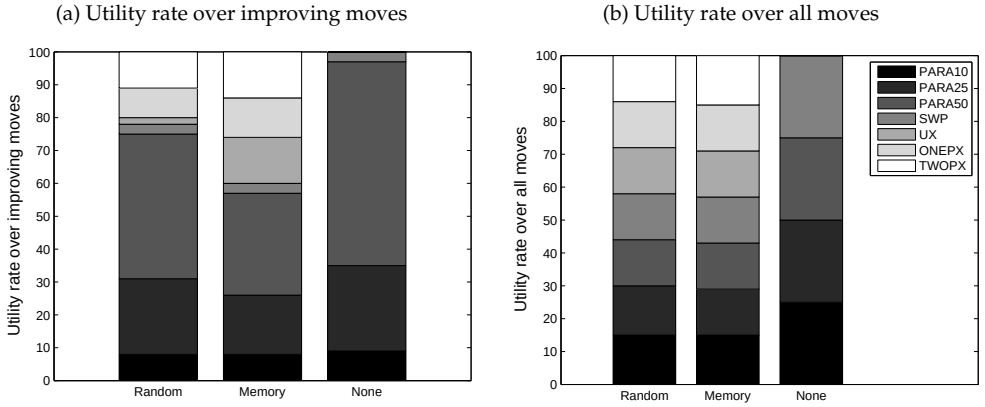


Table 6: Average %-gap over all ORLib instances for each hyper-heuristic using a list of solutions to provide the second child for crossover managed at the domain level

	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** $_{0.76}$

lected overall. Due to 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 of 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 take fractional values as shown in Equation 4 yielding the related LP-relaxed version of the problem. It is widely accepted that the solutions to the LP-relaxed version of the MKP can provide good approximations for the 0-1 version of the problem. Moreover, the framework developed for this work already performs the computational effort required to obtain the solutions to the LP-relaxed problem.

Using the initialisation method for the list of solutions obtained in Section 5.2, the same nine hyper-heuristics are again applied to ORLib using the same parameters as before. Table 6 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 and is 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 difference in statistical significance between these two hyper-heuristics. Interestingly, those hyper-heuristics with Late Acceptance Strategy and Simulated Annealing acceptance outperform those with Only Improving acceptance. This is in contrast to the previous hyper-heuristics which control crossover at the hyper-heuristic level where Only Improving acceptance performed best. As with the previous experiments, the acceptance criteria used has a greater effect on the quality of solutions obtained than selection method. Although there are two 'best' performing hyper-heuristics within this framework we will only compare one hyper-heuristic from each framework in the following section. We will arbitrarily take the Choice Function - Late Acceptance Strategy to compare to the best performing hyper-heuristic from 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 7 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 from Section 6.1 (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 the best performing hyper-heuristic with crossover controlled at the domain-specific level is tested by applying it to two further benchmark sets, each with differing properties. SAC-94 is a set of benchmark instances from classic papers in the literature using mostly real-world data as described in Section 4.4 where optimal solutions are known for each problem. It is difficult to perform a direct comparison with techniques over these instances due to the difference in termination criteria and running times. For example, some methods in the literature provide the best results over 30 runs or more. 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 divided by the number of problems in the set. Choice Function - Late Acceptance Strategy performs a single run on each instance as before. Table 8(a) 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 set of 11 large instances provided by Glover and Kochenberger Glover and Kochenberger (nd). The results for both hyper-heuristics are given as the average of 10 runs on each instance in Table 8(b). The LP-relaxed optimal solutions are again used as a basis to derive %-gap with standard deviations for each instance included as subscript. 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: 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 $_{0.18}$

Table 8: (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)

Dataset	Count	CF-LAS	CF-OI$_{NC}$
hp	2	0.00	0.00
pb	6	0.67	0.50
pet	6	0.50	0.34
sento	2	1.00	1.00
weing	8	0.63	0.63
weish	30	1.00	0.64

(b)

Instance	CF-LAS	CF-OI$_{NC}$
GK01	$0.57_{1.49}$	$1.33_{6.82}$
GK02	$0.81_{3.86}$	$1.60_{9.66}$
GK03	$0.63_{3.10}$	$1.64_{18.25}$
GK04	$0.91_{3.77}$	$1.84_{18.18}$
GK05	$0.45_{3.00}$	$0.83_{13.61}$
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	$0.45_{5.54}$	$0.92_{14.55}$

6.3.1 Comparison to previous approaches

Table 9 shows the results of the best hyper-heuristic presented in this paper, Choice Function - Late Acceptance Strategy with crossover controlled at the domain-specific level compared to a number of techniques from the literature. CPLEX (IBM, 2013) 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 a set of MKP benchmark instances. Here we include results for CPLEX 12.5 on the instances introduced by Chu and Beasley (1998), SAC-94 and the larger benchmarks of Glover and Kochenberger (nd) 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 8GB.

From this table it can be seen that the hyper-heuristics presented in this paper perform well in comparison to many previous approaches. The use of 10^6 fitness evaluations as a termination criteria allows direct comparison to previous metaheuristic approaches (Raidl, 1998; Chu and Beasley, 1998; Özcan and Basaran, 2009; Hinterding, 1994). The %-gap of 0.70 obtained by the hyper-heuristic is better than the previous metaheuristic methods of Özcan and Basaran (2009) and Hinterding (1994) and a number of existing heuristic methods. The best %-gap obtained by a metaheuristic is by the Memetic Algorithm of Chu and Beasley (1998) and the variant of their work provided by Raidl (1998).

The currently best known results in 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 10.

The results of Choice Function - Late 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. Using an independent student's t-test within

Table 9: Average %-gap of other (meta)heuristics and CPLEX over all instances in OR-Lib

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

Table 10: 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*	**0.29** $_{0.03}$	**26**

Table 11: 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

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 10. A fundamental goal of hyper-heuristic research is to provide solutions which are 'good enough, soon enough, cheap enough' (Burke et al., 2003a). 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].

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 11. 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) performs particularly 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 taking a maximum of 0.3 seconds per instance.

Table 12 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 (nd). 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 were allowed 10^6 non-duplicate individuals. Standard deviations for the 30 runs of each instance by Raidl and Gottlieb are provided as subscript. Our hyper-heuristics were also allowed 10^6 evaluations however

[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 3032.5 / 936.47 = 3.24 times faster. t[s]* for Vasquez and Vimont (2005) in Table 10 are scaled using these CPU times.

Table 12: 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	$0.25_{0.01}$	$\mathbf{0.45}_{0.04}$	$0.54_{0.06}$	$0.71_{0.08}$

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 due to the differing nature of each techniques termination criteria and subsequently 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

Two frameworks for controlling crossover in single-point selection hyper-heuristics have been presented using a common NP-hard combinatorial optimisation problem as a testbed. Crossover has been included at two levels, firstly it is controlled at the hyper-heuristic level where no domain-specific information is used, secondly it is controlled below the domain barrier and given domain-specific information. In each case a list of potential second solutions to be used in crossover is 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) has shown to be 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 out performed by simple Only Improving 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 criteria and crossover control scheme has a far greater effect on

solution quality. In the case of domain level crossover control the performance of acceptance criteria is reversed, with Simulated Annealing and Late Acceptance Strategy outperforming Only Improving.

We have introduced a new initialisation scheme for population-based approaches for the MKP which allows the generation of infeasible individuals. This initialisation method was able to outperform two existing initialisation schemes as a methods for providing crossover arguments 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 to allowing infeasible solutions to be used as input for crossover operators. 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 have also been 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 is demonstrated by 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, i.e. mutation heuristics, and 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., editors (2008). *Reactive search and intelligent optimization*. Springer.

Bilgin, B., Özcan, E., and Korkmaz, E. E. (2006). An experimental study on hyper-heuristics and exam timetabling. In Burke, E. K. and Rudová, H., editors, *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2006)*, volume 3867 of *LNCS*, pages 394–412, Brno, Czech Republic. Springer.

Blazewicz, J., Burke, E., Kendall, G., Mruczkiewicz, W., Oguz, C., and Swiercz, A. (2013). A hyper-heuristic approach to sequencing by hybridization of dna sequences. *Annals of Operations Research*. To appear.

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)*, page Extended Abstract, Montreal, Canada.

Burke, E. K., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., and Vazquez-Rodriguez, J. A. (2009a). 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)*, pages 790–797, Dublin, Ireland.

Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P., and Schulenburg, S. (2003a). Hyper-heuristics: An emerging direction in modern search technology. In Glover, F. and Kochenberger, G., editors, *Handbook of Metaheuristics*, chapter 16, pages 457–474. Kluwer Academic Publishers.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. (2010a). Hyper-heuristics: A survey of the state of the art. Technical Report No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. (2010b). *Handbook of Metaheuristics*, chapter A Classification of Hyper-heuristics Approaches, pages 449–468. Springer.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2009b). *Computational Intelligence: Collaboration, Fusion and Emergence*, chapter Exploring Hyper-heuristic Methodologies with Genetic Programming, pages 177–201. 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. (2003b). 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 Raidl, G. R. and Gottlieb, J., editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, volume 3448 of *LNCS*, pages 23–33, Lausanne, Switzerland. Springer.

Chakhlevitch, K. and Cowling, P. (2008). Hyperheuristics: Recent developments. In Cotta, C., Sevaux, M., and Sörensen, K., editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. 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 Raidl, G. R. and Gottlieb, J., editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2005)*, volume 3448 of *LNCS*, pages 34–45, Lausanne, Switzerland. Springer.

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)*, pages 1350–1358, New Orleans, LA, USA. IEEE Press.

Cotta, C. and Troya, J. (1998). *Artificial Neural Nets and Genetic Algorithms*, chapter A Hybrid Genetic Algorithm for the 0-1 Multiple Knapsack Problem, pages 250–254. Springer.

Cowling, P., Kendall, G., and Soubeiga, E. (2001a). A hyperheuristic approach to scheduling a sales summit. In Burke, E. K. and Erben, W., editors, *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2000)*, volume 2079 of *LNCS*, pages 176–190, Konstanz, Germany,. Springer.

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

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.

Denzinger, J., Fuchs, M., and Fuchs, M. (1996). High performance atp systems by combining several ai methods. Technical report, SEKI-Report SR-96-09, University of Kaiserslautern.

Dongarra, J. J. (2013). Performance of various computers using standard linear equations software. Online. retrieved 22/05/2013.

Drake, J. H., Özcan, E., and Burke, E. K. (2012). An improved choice function heuristic selection for cross domain heuristic search. In Coello, C. A. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M., editors, *Proceedings of Parallel Problem Solving from Nature (PPSN 2012), Part II*, volume 7492 of *LNCS*, pages 307–316, Taormina, Italy. Springer.

Fialho, Á., Costa, L. D., Schoenauer, M., and Sebag, M. (2008). Extreme value based adaptive operator selection. In Rudolph, G., Jansen, T., Lucas, S. M., Poloni, C., and Beume, N., editors, *Proceedings of Parallel Problem Solving from Nature (PPSN 2008)*, volume 5199 of *LNCS*, pages 175–184, Dortmund, Germany. Springer.

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

Fleszar, K. and Hindi, K. S. (2009). Fast, effective heuristics for the 0-1 multi-dimensional knapsack problem. *Computers & Operations Research*, 36(5):1602–1607.

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*. W. H. Freeman & Co., New York, NY, USA.

Garrido, P. and Castro, C. (2009). Stable solving of cvrps using hyperheuristics. In Rothlauf, F., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2009)*, pages 255–262, Québec, Canada. ACM.

Glover, F. and Kochenberger, G. (n.d.). Benchmarks for "the multiple knapsack problem". Online. retrieved 03/02/2012.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Boston, MA, USA.

Gottlieb, J. (2000). On the effectivity of evolutionary algorithms for the multidimensional knapsack problem. In Fonlupt, C., Hao, J.-K., Lutton, E., Ronald, E. M. A., and Schoenauer, M., editors, *Proceedings of Artificial Evolution (AE 1999)*, volume 1829 of *LNCS*, pages 23–37, Dunkerque, France. Springer.

Hanafi, S., Lazic, J., Mladenovic, N., Wilbaut, C., and Crevits, I. (2010). New hybrid matheuristics for solving the multidimensional knapsack problem. In Blesa, M. J., Blum, C., Raidl, G. R., Roli, A., and Sampels, M., editors, *Proceedings of the International Conference on Hybrid Metaheuristics (HM 2010)*, volume 6373 of *LNCS*, pages 118–132, Vienna, Austria. Springer.

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)*, volume 4431 of *LNCS*, pages 358–365, Warsaw, Poland. Springer.

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

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*. PhD thesis, University of Nottingham, UK.

IBM (2013). Ibm cplex optimizer. Online.

Jones, T. (1995). Crossover, macromutation, and population-based search. In Eshelman, L. J., editor, *Proceedings of the International Conference on Genetic Algorithms (ICGA 1995)*, pages 73–80, Pittsburgh, PA, USA. Morgan Kaufmann.

Kheiri, A. and Özcan, E. (2013). A hyper-heuristic with a round robin neighbourhood selection. In Middendorf, M. and Blum, C., editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2013)*, volume 7832 of *LNCS*, pages 1–12, Vienna, Austria. Springer.

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)*, pages 188–193, Phoenix, AZ, USA. ACM.

Kiraz, B., Uyar, A. S., and Özcan, E. (2013). Selection hyper-heuristics in dynamic environments. *Journal of the Operational Research Society*. To appear.

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*. The MIT Press, Cambridge, MA.

Lorie, J. and Savage, L. (1955). Three problems in capital rationing. *Journal of Business*, 28(4):229–239.

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., Sttzle, T., and Voss, S., editors (2010). *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, Norwell, MA, USA.

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 Hamadi, Y. and Schoenauer, M., editors, *Proceedings of Learning and Intelligent Optimization (LION 2012)*, volume 7219 of *LNCS*, pages 461–466, Paris, France. Springer.

Moscato, P., Cotta, C., and Mendes, A. (2004). Memetic algorithms. In *New optimization techniques in engineering*. Springer.

Nareyek, A. (2001). *Metaheuristics: computer decision-making*, chapter Choosing Search Heuristics by Non-Stationary Reinforcement Learning, pages 523–544. Kluwer Academic Publishers.

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). Online.

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

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 Runarsson, T. P., Beyer, H.-G., Burke, E. K., Guervós, J. J. M., Whitley, L. D., and Yao, X., editors, *Proceedings of the International Conference on Parallel Problem Solving From Nature (PPSN 2006)*, volume 4193 of *LNCS*, pages 202–211, Reykjavik, Iceland. Springer.

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)*, pages 997–1004, Trondheim, Norway. IEEE Press.

Özcan, E., Misir, M., Ochoa, G., and Burke, E. K. (2010). A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing*, 1(1):39–59.

Petersen, C. (1967). Computational experience with variants of the balas algorithm applied to the selection of r&d projects. *Management Science*, 13(9):736–750.

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 Gottlieb, J. and Raidl, G. R., editors, *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2006)*, volume 3906 of *LNCS*, pages 195–208, Budapest, Hungary. Springer.

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)*, pages 207–211, Anchorage, AK, USA. IEEE Press.

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 Blum, C., Aguilera, M. J. B., Roli, A., and Sampels, M., editors, *Hybrid Metaheuristics*, volume 114 of *Studies in Computational Intelligence*, pages 31–62. Springer.

Rendl, A., Prandtstetter, M., Hiermann, G., Puchinger, J., and Raidl, G. R. (2012). Hybrid heuristics for multimodal homecare scheduling. In Beldiceanu, N., Jussien, N., and Pinson, E., editors, *Proceedings of Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems (CPAIOR 2012)*, volume 7298 of *LNCS*, pages 339–355, Nantes, France. Springer.

Ross, P. (2005). Hyper-heuristics. In Burke, E. K. and Kendall, G., editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Technologies*, chapter 17, pages 529–556. Springer.

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

Swan, J., Özcan, E., and Kendall, G. (2011). Hyperion - a recursive hyper-heuristic framework. In Coello, C. A. C., editor, *Proceedings of Learning and Intelligent Optmization (LION 5)*, LNCS, pages 616–630, Rome, Italy. Springer.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In Schaffer, J. D., editor, *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, Fairfax, Virginia, USA. Morgan Kaufmann.

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 Nebel, B., editor, *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 328–333, Seattle, Washington, USA. Morgan Kaufmann.

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.