Chapter 1

MULTI-OBJECTIVE HYPER-HEURISTIC APPROACHES FOR SPACE ALLOCATION AND TIMETABLING

E.K. Burke, J.D. Landa Silva, E. Soubeiga

School of Computer Science and Information Technology
University of Nottingham, UK
{ekb,jds,exs}@cs.nott.ac.uk

Abstract An important issue in multi-objective optimisation is how to ensure that the obtained non-dominated set covers the Pareto front as widely as possible. A number of techniques (e.g. weight vectors, niching, clustering, cellular structures, etc.) have been proposed in the literature for this purpose. In this paper we propose a new approach to address this issue in multi-objective combinatorial optimisation. We explore hyper-heuristics, a research area which has gained increasing interest in recent years. A hyper-heuristic can be thought of as a heuristic method which iteratively attempts to select a good heuristic amongst many. The aim of using a hyper-heuristic is to raise the level of generality so as to be able to apply the same solution method to several problems, perhaps at the expense of reduced but still acceptable solution quality when compared to a tailor-made approach. The key is not to solve the problem directly but rather to (iteratively) recommend a suitable heuristic chosen because of its performance. In this paper we investigate a tabu search hyper-heuristic technique. The idea of our multi-objective hyper-heuristic approach is to choose, at each iteration during the search, the heuristic that is suitable for the optimisation of a given individual objective. We test the resulting approach on two very different real-world combinatorial optimisation problems: space allocation and timetabling. The results obtained show that the multi-objective hyper-heuristic approach can be successfully developed for these two problems producing solutions of acceptable quality.

Keywords: multi-objective optimisation, Pareto optimisation, hyper-heuristic, local search, diversity preservation.

1. Introduction

In multi-objective optimisation the aim is to find solutions that represent a compromise between the various (sometimes conflicting) criteria used to evaluate the quality of solutions. A solution x is said to be non-dominated with respect to a set of solutions S if there is no other solution in S that is, as good as x in all the criteria and better than x in at least one of the criteria. In Pareto optimisation the goal is to find a set of non-dominated solutions that is representative of the whole trade-off surface, i.e. a non-dominated set that is a good approximation to the Pareto optimal front (Steuer 1986; Rosenthal 1985).

A hyper-heuristic can be viewed as a heuristic that (iteratively) chooses between given heuristics in order to solve an optimisation problem (Burke et al. 2003). One of the main aims of exploring hyper-heuristics is to raise the level of generality at which most current meta-heuristic systems operate. A hyper-heuristic is not concerned with solving a given problem directly as is the case with most meta-heuristic implementations. Instead, a hyper-heuristic solves the problem *indirectly* by recommending *which* solution method (e.g. heuristic) to apply at *which* stage of the solution process. The search is on a heuristic search space rather than a search space of potential problem solutions. One of the motivations is that the same hyper-heuristic method can be applied to a range of problems. For each application problem, our hyper-heuristic only needs a set of heuristics and a formal means for evaluating solution quality (Burke et al. 2003b). The goal is to raise the level of generality of decision support methodology perhaps at the expense of reduced - but still acceptable - solution quality when compared to tailor-made meta-heuristic approaches. Over the past decade or so, hyper-heuristics have been successfully investigated for a number of optimisation problems (e.g. Ayob and Kendall 2004; Burke et al. 2003b; Burke and Newall 2004; Cowling et al. 2000; Cowling et al. 2002; Cowling et al. 2002b; Han and Kendall 2003; Gaw et al. 2004; Ross et al. 2002; Ross et al. 2003). An underlying principle in using a hyper-heuristic approach is that different heuristics have different strengths and weaknesses and it makes sense to try and combine them (the heuristics) in an intelligent manner so that the strengths of one heuristic can compensate for the weaknesses of another (Burke et al. 2003).

This paper proposes the use of hyper-heuristics to help guide the search towards the optimisation of the different individual objectives. This would help to improve the ability of meta-heuristics based on local search to produce non-dominated fronts to better approximate the Pareto front. By using a hyper-heuristic approach, a heuristic is chosen in order to guide the search towards the desired regions of the trade-off surface. This strategy takes into consideration the localization of the current solution(s) in the objective space and the ability of each neighbourhood exploration heuristic to achieve improvements on each of the individual objectives. That is, a hyper-heuristic systematically tries to apply the neighbourhood exploration heuristic that improves on 'poor' objectives while maintaining the quality of 'rich' objectives on a given solution. This is a novel approach for tackling the problem of achieving a good coverage of the desired trade-off surface in multi-objective combinatorial optimisation.

2. Techniques for Improving the Distribution of Non-dominated Sets

Among the meta-heuristics proposed for Pareto optimisation there are single-solution approaches and population-based approaches (Coello Coello et al. 2002; Deb 2001; Jones et al. 2001). One of the issues of major concern when developing meta-heuristics for Pareto optimisation is how to ensure that the algorithm produces a uniformly distributed non-dominated set of solutions at the end of the search. Researchers have shown the importance of maintaining a good distribution of solutions along the trade-off surface for the good performance of population-based meta-heuristics for Pareto optimisation (e.g. Laumanns et al. 2001). Several strategies that aim to improve the distribution of non-dominated solutions have been proposed. For example, the search can be directed towards the desired area of the trade-off surface by tuning weights (e.g. Czyzak and Jaszkiewicz 1998; Ishibuchi et al. 2002; Ulungu et al. 1999). Clustering or niching methods attempt to achieve a good distribution by assigning fitness to solutions based on the density of solutions in a given area (e.g. Knowles and Corne 2000; Lu and Yen 2002; Socha and KisielDorohinicki 2002). Fitness sharing is a clustering technique that reduces the fitness of solutions in proportion to the number of solutions that are close together (e.g. Horn 2003; Talbi et al. 2001; Zhu and Leung 2002). Cellular structures and adaptive grids are also clustering techniques that aim to uniformly distribute the solutions over the trade-off surface (Murata et al. 2001; Toscano Pulido and Coello Coello 2003). Restricted mating sets the probability of recombining two solutions according to the degree of similarity between these solutions in order to avoid the generation of new solutions that are 'too-similar' (Ishibuchi and Shibata 2003; Kumar and Rockett 2002). Relaxed forms of the dominance relation (e.g. Burke and Landa Silva 2002; Burke and Landa Silva 2005; Deb et al. 2003; Laumanns et al. 2002; Mostaghim and

Teich 2003; Jin and Wong 2003) and entropy metrics (Gunawan et al. 2003) have also been proposed to improve the ability of multi-objective meta-heuristics to achieve a good coverage of the trade-off surface. Also, fuzzy logic has been used to provide different degrees of Pareto optimality within non-dominated sets (Farina and Amato 2003).

Most of the above techniques attempt to 'restrict' the likelihood of generating solutions in 'crowded' regions of the trade-off surface and 'boost' the likelihood of generating solutions in 'under-populated' regions. From these techniques, the specification of the search direction by tuning weights is the method that directly attempts to 'push' the current solution(s) towards the desired region of the trade-off surface. The hyper-heuristic approach proposed here follows this same strategy, but attempts to do it in a more 'intelligent' way by applying the neighbourhood search heuristic that is more likely to 'push' the solution in the desired direction.

3. A Multi-objective Hyper-heuristic Approach

In this section we first give a brief overview of the work presented in the literature related to hyper-heuristics. Then, we describe our proposed hyper-heuristic approach for multi-objective combinatorial optimisation. Next, we develop four multi-objective hyper-heuristic algorithms based on the tabu search framework proposed by (Burke et al. 2003b) for single-objective optimisation.

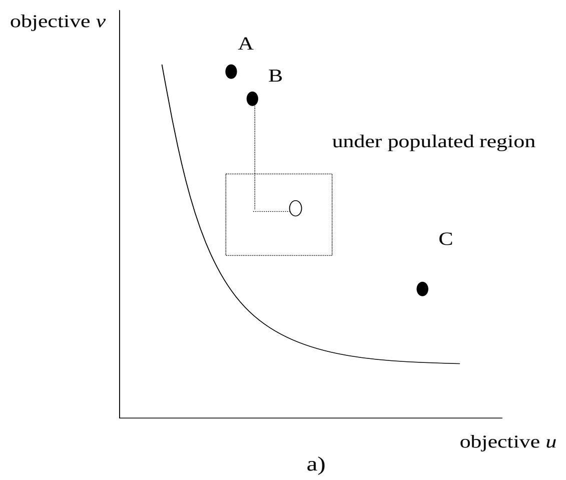
Related Work

It has been shown that the use of various 'simple' neighbourhood heuristics can be beneficial when tackling complex combinatorial optimisation problems. For example, variable neighbourhood search is based on the systematic change of the neighbourhood structure during the search (Hansen and Mladenovic 2001). Salman et al. proposed a cooperative team of heuristics to generate non-dominated solutions for the two-objective sparse knapsack problem (Salman et al. 2002). In that approach, an asynchronous architecture was implemented in which a number of heuristics (constructors, improvers and destroyers) evolve a shared population of solutions. The heuristics work in an asynchronous fashion and each one decides when to work, on which solutions to work, and how to generate or improve solutions.

Hyper-heuristics are designed to control the application of a set of heuristics during the search process (Burke et al. 2003). At each time during the search, the selection of the next heuristic to be used is based on the past performance that each of them has exhibited. Note that an important feature of hyper-heuristics is that the set of heuristics that are applied during the search can be simple neighbourhood exploration heuristics (as in Hansen and Mladenovic 2001 and Salman et al. 2002) or more elaborate algorithms such as meta-heuristics. The idea of using hyper-heuristics is that the resulting method should be able to produce solutions which are 'soon-enough, good-enough, cheap enough', while remaining competitive with problem specific techniques. Although the term hyper-heuristic has been proposed in recent years (see Burke et al. 2003), a number of these approaches have been developed particularly in the past ten years or so. For example, Hart and Ross used a genetic algorithm based hyper-heuristic to solve the job-shop scheduling problem (Hart and Ross 1998). In their approach, the chromosome represents which method to use in order to identify conflicts amongst schedulable operations and which heuristic to use in order to select an operation from the conflicting sets. Cowling et al. applied a genetic algorithm based hyper-heuristic to tackle the trainer scheduling problem in which the chromosome represents the ordered sequence of simple heuristics to be applied during the search process (Cowling et al. 2002). The application of hyper-heuristics using a simple choice function to rank a set of simple heuristics has been reported in (Cowling et al. 2000; Cowling et al. 2001; Cowling et al. 2002).

The Proposed Approach

The multi-objective hyper-heuristic approach proposed here is based on a tabu-search hyper-heuristic method which was developed in (Burke et al. 2003b) for single-objective optimisation. The performance of a given neighbourhood exploration heuristic may depend on the problem domain, the particular instance of the problem domain and the present conditions of the search process. For some multi-objective combinatorial optimisation problems, a set of simple neighbourhood exploration heuristics can be developed fairly quickly. Then, the approach proposed here selects the most appropriate neighbourhood heuristic at certain points during the search in order to 'push' the solution in the desired direction towards the Pareto optimal front. An 'intelligent' way to do this is by 'learning' how well each simple heuristic achieves improvements on each of the objectives of a given solution. Having a hyper-heuristic that systematically chooses the best strategy to explore the neighbourhood of the current solution(s) can help to obtain a uniformly distributed set of non-dominated solutions. The 'learning' mechanism can be implemented so as to 'reward' well-performing heuristics and 'punish' badly-performing ones. The hyper-heuristic therefore maintains a list of the



a)

		improves v		**deteriorates v**	
		small	large	small	large
improves u	small	H_2 H_3	-----	-----	H_6
	large	-----	H_1 H_8	-----	H_7
deteriorates u	small	-----	H_5	-----	-----
	large	-----	H_4	-----	-----

b)

Figure 1.1. a). Towards a better coverage of the trade-off surface. Solutions in crowded regions of the trade-off surface (such as solution B) are pushed towards the under-populated regions (such as the region enclosed by the rectangle). b). Directed neighbourhood search for a better coverage of the trade-off surface. Each heuristic might produce an improvement or detriment on each particular objective. Then, the adequate heuristics can be applied to improve upon specific objectives and hence, to push the solution from one region of the trade-off surface to another region.

heuristics and controls their application during the search following the principles of tabu search (Glover and Laguna 1997). The tabu search hyper-heuristic measures the performance of each simple neighbourhood exploration heuristic and adapts it according to the knowledge gained during the search and during previous runs of the algorithm. The aim of this adaptation is to approximate the trade-off surface in a more efficient way by using those moves that are more promising according to the current quality of the various objectives and the historical knowledge about the search process.

The idea described above is illustrated in Fig. 1.1, where a two-objective minimisation problem is considered. The desired trade-off surface and three non-dominated solutions are shown. Solutions A and B are ‘good’ with respect to the objective u but are ‘bad’ with respect to the objective v. On the other hand, solution C is 'good' with respect to objective v but it is 'bad' with respect to objective u. The region of the trade-off surface enclosed in the rectangle can be considered to be under-populated because no solutions have been found in that area. Figure 1.1 shows how each of eight neighbourhood heuristics could perform with respect to each objective. Then, in order to obtain a better coverage of the trade-off surface, the following strategy can be used: maintain solutions A and C in their current locations and 'push' solution B towards the under-populated region. This can be achieved by applying a heuristic which yields a large improvement on objective v (possibly with a small deterioration on objective u). Heuristics H_1, H_8, and H_5 of Fig. 1.1 can be considered good candidate heuristics to achieve this. The challenge for the hyper-heuristic is to choose the right heuristic for the right operation at the right time during the search. It should be noted that the hyper-heuristic thus operates in the *heuristic space* as opposed to most implementations of meta-heuristics which operate in the *solution space.*

The Tabu Search Hyper-heuristic Framework

The basic idea of the tabu search hyper-heuristic framework was initially introduced in (Burke et al. 2003b). In this framework, the heuristics can be thought of as competing with one another. The competition rules are inspired from the principles of reinforcement learning (Kaelbling et al. 1996; Sutton and Barto 1998). At the beginning of the search each heuristic k has a score of 0 points (i.e. $r_k = 0$). These scores are allowed to decrease and increase - within the interval $[r_{min}, r_{max}]$, where r_{min}, r_{max} are respectively the lower and upper rankings - to reflect the performance of the corresponding heuristics. Let Δ represent the change in the objective function value from the previous solution to the new one. If the application of a given heuristic results in an improvement of the current solution (i.e. $\Delta > 0$) then the score of the heuristic is increased, e.g. $r_k = r_k + \alpha$. Otherwise it is decreased, e.g. $r_k = r_k - \alpha$, where α is a positive number. There are several ways to choose α. Here we choose $\alpha = 1$ (see Burke et al. 2003b).

In addition to this ranking scheme, a tabu list of heuristics is maintained, which excludes certain heuristics from the competition for a certain duration. The basic idea of the tabu list is to prevent a heuristic which did not perform well from being chosen too soon (even if it has the highest rank). More precisely, we include heuristic k in the tabu list (on a 'First In - First Out' basis) if Δ is non-positive. Furthermore, heuristics already in the tabu list are released if Δ is positive. The idea is that there is no point in keeping a heuristic tabu once the current solution has been improved (e.g. when $\Delta \geq 0$). Thus we employ a variable-length dynamic tabu list of heuristics. The basic tabu search hyper-heuristic can be outlined by the following pseudocode:

```
Do
1- Select the heuristic, k, with highest rank and apply it.
2- If $\Delta > 0$ then $r_k = r_k + \alpha$ and empty TABULIST.
3- Else $r_k = r_k - \alpha$ and include k in TABULIST.
Until Stopping condition is met.
```

Multi-objective Tabu Search Hyper-heuristic Algorithms

In this paper we propose to adapt the above single-objective hyper-heuristic approach to multi-objective optimisation. Consequently, the following modifications are made:

1 The performance of each heuristic is no longer evaluated with respect to a single objective (or aggregate objective) but instead with respect to individual objectives. This implies that Δ is replaced with Δ_u and r_k is replaced with $r_k(u)$ where $u = 1, 2, ...q$ and q is the number of objectives in the problem.

2 As a result of the above change, we also have to decide (choose) which individual objective to deal with at any one time.

3 A third level of design concerns the tabu list. It is obvious that instead of one tabu list, we may now have several (i.e. one tabu list for each of the q objectives).

We implemented three algorithms with different combinations of the above three modifications. Within the three algorithms, heuristic performance is evaluated with respect to each objective.

Single Tabu Random Uniform (TSRandUnif). In this algorithm the individual objective is chosen uniformly at random. Here, there is only one tabu list. The algorithm works as shown in the pseudocode below. The *on-line non-dominated set* contains all the non-dominated solutions that are obtained in each iteration of step 2. That is, for each solution in the initial population. The *off-line non-dominated set* contains the non-dominated solutions obtained from the P *on-line non-dominated sets.*

```
1- Randomly generate an initial population of $P$ solutions.
2- For each solution in the initial population, Do
  2.1- Select an individual objective $u$ uniformly at random.
  2.2- Select the heuristic, $k$, with highest rank $r_k(u)$ and apply it to the current solution.
  2.3- If $\Delta_u > 0$ then $r_k(u) = r_k(u) + \alpha$ and empty TABULIST.
  2.4- Else $r_k(u) = r_k(u) - \alpha$ and include $k$ in TABULIST.
  2.5- For all other individual objectives $v = 1, 2, ...q$ and $v \neq u$, Do
    2.5.1- If $\Delta_v > 0$ then $r_k(v) = r_k(v) + \alpha$.
    2.5.2- Else $r_k(v) = r_k(v) - \alpha$.
  2.6- Update the on-line set of non-dominated solutions.
3- Until Stopping condition is met.
4- Generate the off-line set of non-dominated solutions.
```

Single Tabu Roulette-Wheel (TSRoulWheel). In this algorithm, the choice of individual objectives is based on roulette-wheel selection (see Goldberg 1989). A given individual objective u is chosen with a probability that is proportional to the distance from the value of u in the current solution to the value of u in an ideally optimal solution (i.e. a solution in which the value of each objective is optimal - such a solution may not exist.). The idea here is that the worse the value of an individual objective (relative to the others), the higher the probability of that objective being chosen by the roulette-wheel selection. Of course, the sum of all probabilities over all the individual objectives must be equal to 1. As in the previous algorithm, here too, there is only one tabu list. The algorithm works as follows:

```
1- Randomly generate an initial population of $P$ solutions.
2- For each solution in the initial population, Do
  2.1- Select individual objective $u$ using roulette wheel selection.
  2.2- Select the heuristic, $k$, with highest rank $r_k(u)$ and apply it to the current solution.
  2.3- If $\Delta_u > 0$ then $r_k(u) = r_k(u) + \alpha$ and empty TABULIST.
  2.4- Else $r_k(u) = r_k(u) - \alpha$ and include $k$ in TABULIST.
  2.5- For all other individual objective $v = 1, 2, ...q$ and $v \neq u$, Do
    2.5.1- If $\Delta_v > 0$ then $r_k(v) = r_k(v) + \alpha$.
    2.5.2- Else $r_k(v) = r_k(v) - \alpha$.
  2.6- Update the on-line set of non-dominated solutions.
3- Until Stopping condition is met.
4- Generate the off-line set of non-dominated solutions.
```

Multiple Tabu Roulette-Wheel (MTSRoulWheel). In this algorithm the choice of individual objectives is based on roulette-wheel selection as in the previous one (TSRoulWheel). However, this algorithm maintains multiple tabu lists. In effect, there is one tabu list for each objective $u = 1, 2, ..., q$. The algorithm works as follows:

```
1- Randomly generate an initial population of P solutions.
2- For each solution in the initial population, Do
   2.1- Select individual objective u using roulette wheel selection.
   2.2- Select the heuristic, k, with highest rank $r_k(u)$ and apply it to the current solution.
   2.3- For each objective $u = 1, 2, ...q$, Do
      2.3.1- If $\Delta_u > 0$ then $r_k(u) = r_k(u) + \alpha$ and empty TABULIST(u).
      2.3.2- Else $r_k(u) = r_k(u) - \alpha$ and include k in TABULIST(u).
   2.4- Update the on-line set of non-dominated solutions.
3- Until Stopping condition is met.
4- Generate the off-line set of non-dominated solutions.
```

It can be seen from the above procedures that the main difference between *TSRandUnif* on the one hand and *TSRoulWheel* and *MTSRoulWheel* on the other hand is in step 2.1, which is a simple random uniform selection for the former and a roulette wheel selection for the two latter algorithms. The difference between *TSRoulWheel* and *MTSRoulWheel* is in the number of tabu lists used.

Pure Random (PureRand). In order to investigate if there is a benefit in incorporating a learning mechanism (which, in this case, consists of the heuristic ranking system, the tabu list of heuristics and the objective selection mechanism) into our hyper-heuristics, we implemented one algorithm in which the learning is disabled. In the complete absence of a learning mechanism the choice of a heuristic is simply made randomly. The resulting algorithm repeatedly chooses one heuristic uniformly at random and applies it once. This simple algorithm is illustrated in the following pseudocode.

```
1- Randomly generate an initial population of P solutions.
2- For each solution in the initial population, Do
   2.1- Select a heuristic uniformly at random and apply it once to the current solution.
   2.2- Update the on-line set of non-dominated solutions.
3- Until Stopping condition is met.
4- Generate the 'off-line' set of non-dominated solutions.
```

Note that, like the tabu search hyper-heuristics, *PureRand* maintains an *on-line non-dominated set* of solutions. It also produces the *off-line non-dominated set* of solutions, which is the output of the algorithm.

In the next two sections we report results that are obtained by applying the above hyper-heuristic approaches to two different real-world combinatorial optimisation problems. As already mentioned, in order to apply a hyper-heuristic to a given problem, all that is needed is a set of simple neighbourhood search heuristics and a means of evaluating solution quality. These will be given for each problem considered below. It should be noted that our multi-objective hyper-heuristic approaches are not designed with a particular problem in mind. On the contrary, the goal is to develop an approach which is more general than current meta-heuristic approaches to multi-objective optimisation. Moreover, the only mechanism used to obtain a good distribution of solutions over the trade-off surface is the learning mechanism incorporated into the hyper-heuristics. We show below that our approach is both effective and general in terms of the two problems considered in this paper.

4. Application to Space Allocation

In this section we present the application of the multi-objective hyper-heuristic approaches described above to the space allocation problem. This problem refers to the distribution of office space in academic institutions. First, we give a description and formulation of the problem. Then, we describe the heuristics employed to carry out the neighbourhood search. This is followed by a presentation and discussion of the results obtained in our computational experiments.

Problem description

The space allocation problem is a difficult real-world combinatorial optimisation problem that is closely related to the class of knapsack problems (Martello and Toth 1990). The particular space allocation problem considered here is the distribution, in an academic institution, of the available room space among a set of entities (staff, research students, computer rooms, lecture rooms, etc.) in such a way that the misuse of room space and the violation of soft constraints are minimised. Soft constraints are restrictions that limit the ways in which the entities can be allocated to the rooms (e.g. entities that should not share a room, entities that should be allocated together, etc.) and that are penalised if violated. The following types of constraints exist in the problem instances considered in this paper:

1 *Not Sharing* - two entities cannot share a room (e.g. professors must have private offices). A penalty of 50 is applied if a constraint of this type is violated.

2 *Be located in* - a given entity should be allocated to a given room (e.g. a computer lab). A penalty of 20 is applied if a constraint of this type is violated.

3 *Be adjacent to* - two given entities should be allocated in adjacent rooms (e.g. a PhD student and his supervisor). A penalty of 10 is applied if a constraint of this type is violated.

4 *Be away from* - two given entities should be allocated away from each other (e.g. lecture room and photocopier room). A penalty of 10 is applied if a constraint of this type is violated.

5 *Be together with* - two given entities should be allocated in the same room (e.g. two PhD students working on the same project). A penalty of 10 is applied if a constraint of this type is violated.

6 *Be grouped with* - a given entity should be allocated in a room that is 'close' to a given set of entities (e.g. the members of a research group). A penalty of 5 is applied if a constraint of this type is violated.

When a particular constraint is not *soft* but *hard*, it must be satisfied for the solution to be considered feasible. Depending on the problem instance, any of the above types of constraints can be *hard* or *soft*.

More formally, the space allocation problem refers to the allocation of a set of n entities into a set of m available rooms. Each entity $j = 1, 2, ..., n$ has a space requirement $w(j)$. Similarly, each room $i = 1, 2, ..., m$ has a capacity $c(i)$. Each entity must be allocated to exactly one room and each room can contain zero or more entities. The aggregated space requirements of all the entities allocated to a room i is denoted $Q(i)$. For a given room i, there is space wastage if $c(i) > Q(i)$ and there is space overuse if $c(i) < Q(i)$. There is a penalty of 1 for each unit of space wasted and a penalty of 2 for each unit of space overused (it is less desirable to overuse space than to waste it). The sum of penalties due to space wastage and to overused space for all m rooms is called space misuse and is denoted by $F1$. The sum of all penalties due to the violation of soft constraints is denoted by $F2$. This problem is tackled as a two-objective optimisation problem in this paper, where $F1$ and $F2$ are minimisation objectives.

A solution or allocation is represented by a vector π of length n where each element $\pi(j) \in 1, 2, .., m$ for $j = 1, 2, ..., n$ indicates the room to which the entity j is allocated. For a more detailed description of the space allocation problem see (Burke and Varley 1998; Landa Silva 2003).

Neighbourhood Search Heuristics

Several neighbourhood exploration heuristics have been designed based on three moves: *relocate*, *swap* and *interchange.* A *relocate* move changes one entity from one room to another. In a *swap* move, the assigned rooms between two entities are swapped. The third move *interchanges* all the entities between two rooms. That is, there are three neighbourhood structures, one defined by each type of move. However, there can be many ways in which to explore each of these neighbourhood structures and each of these is a neighbourhood exploration heuristic. In our implementation, there are three neighbourhood structures and nine neighbourhood exploration heuristics, three for each neighbourhood structure. These neighbourhood exploration heuristics are outlined below.

RelocateRndRnd : Selects an allocated entity and room at random and relocates the entity to the chosen room.

RelocateRndBestRnd : Selects an allocated entity at random. Next, explores a number of randomly selected rooms evaluating the suitability of each of them to relocate the selected entity. Then, the chosen entity is relocated to the best of the subset of explored rooms.

RelocatePenaltyBestRnd : The allocated entities are sorted in non-increasing order of their individual penalties (violation of soft constraints). In each iteration, the allocated entity with the highest penalty is selected and the room to relocate this entity is chosen with the same procedure as in RelocateRndBestRnd.

SwapRndRnd : Selects two allocated entities at random and makes the swap move.

SwapRndBestRnd : Selects an allocated entity at random. Next, explores a number of randomly selected allocated entities evaluating the suitability of each of them to be swapped with the other entity. Then, the best of the subset of explored entities is chosen to make the swap move.

SwapPenaltyBestRnd : The allocated entities are sorted in non-increasing order of their individual penalties (violation of soft constraints). The allocated entity with the highest penalty is selected and the other entity to make the swap is chosen with the same procedure as in SwapRndBestRnd.

InterchangeRndRnd : Selects two rooms at random and makes the interchange move.

InterchangeRndBestRnd : Selects a room at random. Next, it explores a number of randomly selected rooms and evaluates the suitability of each of them to make the interchange. Then, the best of the subset of explored rooms is chosen to implement the interchange move.

InterchangePenaltyBestRnd : The rooms are sorted in non-increasing order of their individual penalties (space misuse and violation of soft constraints). The room with the highest penalty is selected and the room to make the interchange is chosen with the same procedure as in InterchangeRndBestRnd.

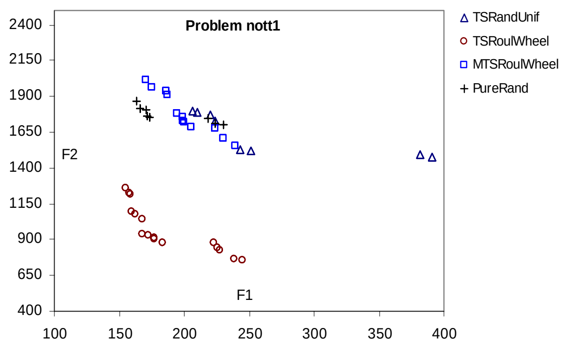
Computational results

Experimental Settings. All algorithms were coded in Microsoft Visual C++ version 6, and all experiments were run on a PC Pentium III 1000MHz with 128MB RAM running on Microsoft Windows 2000. Three problem instances: nottl, nott1b and trent1 were used in these experiments. The nott1 and nott1b test intances were prepared using real data corresponding to the distribution of office space in the School of Computer Science and Information Technology at the University of Nottingham during the 1999-2000 academic year. In the nott1 instance there are 131 rooms, 158 entities to be allocated, and 263 constraints (111 hard and 152 soft). The nott1b instance has 115 rooms, 142 entities, and 260 constraints (110 hard and 150 soft). The trent1 instance was prepared using real data corresponding to the distribution of office space in the Chaucer bulding at the Nottingham Trent University during the 2000-2001 academic year. In the trent1 instance there are 73 rooms, 151 entities, and 211 constraints (80 hard and 131 soft). For full details of these data sets see `www.cs.nott.ac.uk/~jds/research/spacedata.html`.

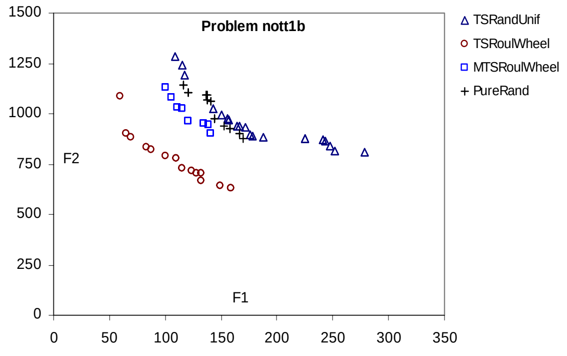
For each test instance, a population of size 20 was generated as follows. One entity is selected at random. Then, the best of a subset of randomly selected rooms is chosen to allocate the entity, ensuring that no hard constraint is violated. This process is repeated for each entity until all of them are allocated to a room. Each of the four hyper-heuristic algorithms described in section 1.3.0 was applied to each of the problem instances. The termination condition for each algorithm was set to a maximum number of solution evaluations *eval* (i.e. *eval*/*P* evaluations for each solution in the population). The value of *eval* was set to 100000, 80000 and 50000 for nott1, nott1b and trent1 respectively. The off-line non-dominated sets obtained by the algorithms are shown in Fig. 1.2.

Comparing the Hyper-heuristic Approaches. It is clearly observed in Fig. 1.2 that for the three problem instances, the *TSRoulWheel* hyper-heuristic produces the best non-dominated sets. This algorithm seems to be particularly good when applied to the problem nott1 because each of the solutions produced by this algorithm dominates all the solutions obtained by the other three approaches. For the problems nott1b and trent1, the *TSRoulWheel* algorithm produces one or more solutions that dominate each single one of the solutions obtained with the other algorithms. Furthermore, *TSRoulWheel* achieves a good coverage of the trade-off front in the three problem instances while the non-dominated solutions produced by the other three algorithms produce solutions that are clustered in some region on the trade-off front. That is, from these results, it can be visually verified that the best non-dominated fronts for the three problem instances are obtained with the *TSRoulWheel* hyper-heuristic. There is not a clear ranking when comparing the performances of the other three algorithms. Only in the instance trent1 does *TSRandUnif* clearly outperform *MTSRoulWheel* and *PureRand*. It can also be observed in Fig. 1.2 that the learning mechanism incorporated into the *TSRoulWheel* algorithm (the combination of the heuristic ranking system, the tabu list of heuristics and the objective selection mechanism) seems to be effective in managing the set of simple neighbourhood search heuristics for obtaining good sets of non-dominated solutions. From the other three algorithms, *PureRand* does not incorporate any element of the learning mechanism and *TSRandUnif* does not incorporate the roulette-wheel mechanism to select the objective. In the case of *MTSRoulWheel*, it seems that having two lists of heuristics deteriorates the performance of the learning mechanism. The mechanism implemented in *MTSRoulWheel* to manage one tabu list for each objective (two in this problem) is a basic one, it simply associates one tabu list to each objective. We are currently investigating more sophisticated mechanisms to manage the tabu list(s) of heuristics.

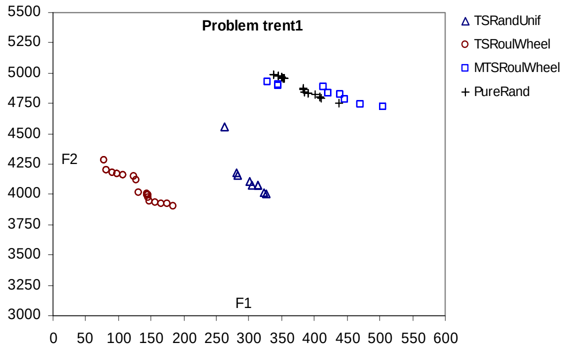
Comparison with a Population-based Annealing Algorithm. A population-based annealing algorithm (*PBAA*) has been tailored for the space allocation problem (Burke and Landa Silva 2005). This algorithm is fairly sophisticated and a considerable amount of work has been invested in its design. This approach is a hybrid algorithm that evolves a population of solutions using a local search heuristic H_{LS} and a



a)



b)

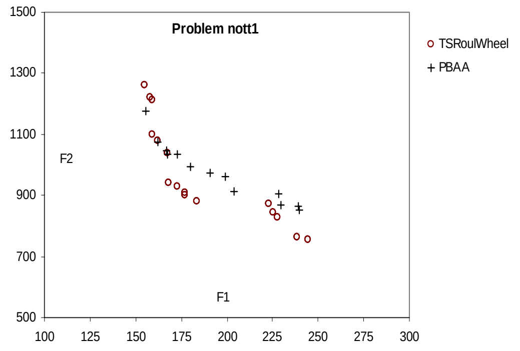


c)

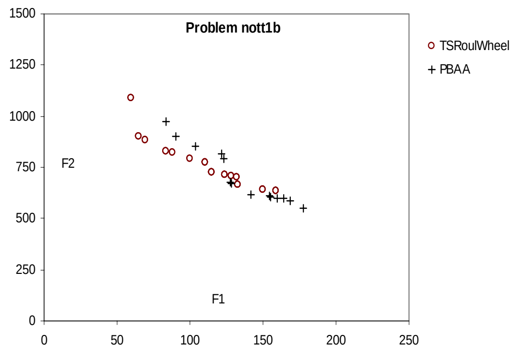
Figure 1.2. Non-dominated sets obtained with each of the hyper-heuristics for the problems a)nott1 b)nott1b and c)trent1.

mutation operator. This local search heuristic manages the same simple neighbourhood search heuristics described above for the space allocation problem. However, H_{LS} incorporates knowledge of the problem domain to decide which neighbourhood search heuristic to apply according to the status of the current solution. The mutation operator disturbs a solution in a controlled way by removing from their assigned room those entities that have the highest penalties. These entities are then re-allocated to different rooms in an attempt to diversify the search. A common annealing schedule controls the evolution of the whole population and a cooperation mechanism is incorporated in order to encourage the sharing of good parts of solutions among individuals in the population and hence, to avoid the exploration of already visited bad solutions. A more detailed description of this population-based annealing algorithm can be seen elsewhere (Burke and Landa Silva 2005).

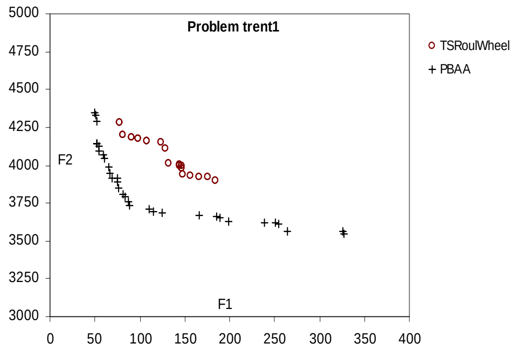
The non-dominated fronts obtained by the *TSRoulWheel* algorithm above are compared with those produced by the *PBAA* approach in Fig. 1.3. The results for *PBAA* were reported in (Burke and Landa Silva 2005) and were obtained using the same termination condition used in this paper and on the same computer. The results shown in Fig. 1.3 show that none of the two algorithms appears to clearly outperform the other one. For example, in problem nott1, the *TSRoulWheel* produces a better front. In the problem nott1b, *TSRoulWheel* outperforms *PBAA* in the upper part of the trade-off surface while *PBAA* does better in the lower part of the front. In the problem trent1, the *PBAA* algorithm clearly obtains better results than *TSRoulWheel*. We have shown in (Burke and Landa Silva 2005) that knowledge of the problem domain incorporated into the *PBAA* approach helps to obtain high quality sets of non-dominated solutions. Here, we can see that the fairly simple *TSRoulWheel* hyper-heuristic approach appears to be competitive. As noted above, we do not expect the *TSRoulWheel* hyper-heuristic approach to produce better solutions than a well-tuned algorithm that incorporates knowledge of the problem domain. But, as we show here, this multi-objective hyper-heuristic is easy to implement and produces acceptable results. In the next section, we apply the four hyper-heuristics to the university course timetabling problem. We aim to demonstrate that these approaches are not only effective but that they can also be readily applied to different problems with solution quality still being competitive.



a)



b)



c)

Figure 1.3. Non-dominated sets obtained by the TSRoulWheel hyper-heuristic and the tailor-made Population-based Annealing Algorithm (PBAA) for the problems a)nott1 b)nott1b and c)trent1.

5. Application to Timetabling

In this section we present the application of the multi-objective hyper-heuristic approaches described above to the course timetabling problem. This problem refers to the scheduling of a set of events to a time period while satisfying a number of constraints. First, we give a description and formulation of the problem. Then, we describe the heuristics employed to carry out the neighbourhood search. This is followed by a presentation and discussion of the results obtained in our computational experiments.

Problem description

In the university course timetabling problem the aim is to schedule a number of events such as lectures, seminars, tutorials, etc. in the available timeslots and satisfying a number of additional constraints (see Burke et al. 1997; Carter:Laporte 1998; Schaerf 1999). The problem instances used in this paper are taken from the literature (Rossi-Doria et al. 2003; Socha et al. 2002). L is the set of events to be scheduled. There are 5 days and in each day there are 9 hourly timeslots, that is, there are 45 available timeslots in total. R is the set of rooms in which events can take place. S denotes the set of students who attend the events. There is also a set F of features satisfied by rooms and required by events (e.g. event e requires a room equipped with an overhead projector, or a sound system, video conference facilities etc.). Each student is required to attend a number of events. Each room has a maximum seating capacity. The following constraints are considered to be hard (must be satisfied):

1 No student can attend more than one event in the same timeslot.

2 The room in which an event takes place, satisfies all the features required by the event.

3 The capacity of the room cannot be exceeded.

4 At most one event is scheduled in the same combination of room and timeslot.

There are also a number of additional constraints that should be satisfied whenever possible. If any of these constraints is violated, a penalty is applied to the solution. The soft constraints (desirable to be satisfied) are listed below:

1 A student has a scheduled event in the last timeslot of the day.

2 A student should attend more than 2 consecutive events.

3 A student has only one event to attend on a given day.

Burke et al. used $E = 1000 \times Hcv + Scv$, where Hcv is the number of hard constraint violations and Scv the number of soft constraint violations, to evaluate solution quality (Burke et al. 2003b). Here, we tackle the problem in a multi-objective fashion. Feasible solutions (ones which satisfy the hard constraints) are those for which $Hcv = 0$. Each feasible solution is then evaluated using the following three objectives to be minimised: LS (respectively EiR and SC) counts the number of violations of the first (respectively second and third) soft constraint.

We represent solutions in the same way as in (Rossi-Doria et al. 2003; Socha et al. 2002). A timetable is represented using a vector of length $|L|$. Each position in the vector corresponds to one event. That is, position j corresponds to event e_j for j = 1,...,$|L|$. In each position of the vector, there is an integer number in the interval $[1, 45]$ that indicates the timeslot in which the corresponding event has been scheduled. For example, in the vector: [39, 10, ..., 45], event e_1 is scheduled in timeslot 39, e_2 is scheduled in timeslot 10, ..., and event $e_{|L|}$ is scheduled in timeslot 45. Similarly to (Rossi-Doria et al. 2003; Socha et al. 2002), in this paper we also tackle the problem of room assignment in a separate way by means of a matching algorithm. This algorithm is applied to the solution every time the solution is modified by any of the neighbourhood search heuristics. These heuristics are described next.

Neighbourhood Search Heuristics

We used eight neighbourhood search heuristics which are all simple and based on the neighbourhood moves described in (Rossi-Doria et al. 2003; Socha et al. 2002). These heuristics are described as follows:

h1 : Select an event at random and move it from its current timeslot to a different timeslot selected at random too. This move is performed only if $Hcv > 0$ (i.e. solution is infeasible).

h2 : Select an event at random and move it from its current timeslot to a different timeslot selected at random too. This move is performed only if $Hcv = 0$ (i.e. solution is already feasible) and the move does not result in $Hcv > 0$.

h3 : Swap the timeslots of two events selected at random. This move is performed only if $Hcv > 0$ (i.e. solution is infeasible).

h4 : Swap the timeslots of two events selected at random. This move is performed only if $Hcv = 0$ (i.e. solution is already feasible) and the move does not result in $Hcv > 0$.

h5 : Same as the heuristic [h1] but the timeslot selected for the move is the first one that provokes an improvement on Hcv.

h6 : Same as the heuristic [h2] but the timeslot selected for the move is the first one that provokes an improvement on Scv.

h7 : Same as the heuristic [h3] but the pair of timeslots selected for the swap is the first one that provokes an improvement on Hcv.

h8 : Same as the heuristic [h4] but the pair of timeslots selected for the swap is the first one that provokes an improvement on Scv.

where $Scv = LS + EiR + SC$ is the aggregate objective.

Computational Results

Experimental Settings. All algorithms were coded in Microsoft Visual C++ version 6, and all experiments were run on a PC Pentium III 1000MHz with 128MB RAM running on Microsoft Windows 2000. To test the effectiveness of our hyper-heuristic approaches we considered problem instances taken from (Socha et al. 2002). A set of 20 initial feasible solutions were randomly obtained using the basic tabu search hyper-heuristic of (Burke et al. 2003b). To allow for a fair comparison, all four algorithms described above in this paper (*TSRandUnif*, *TSRoulWheel*, *MTSRoulWheel* and *PureRand*) start from the same initial set of solutions. Each of the 20 initial solutions is generated using the same algorithm of (Burke et al. 2003b) with different random seeds.

Comparing the Hyper-heuristic Approaches. We show in Fig. 1.4 to 1.6 the off-line non-dominated sets obtained by all four algorithms when applied to three problem instances of medium size (400 events, 10 rooms and 5 features) described in (Socha et al. 2002). The stopping condition is 1000 iterations for each individual solution, i.e. 20000 iterations for the whole population. Of the four algorithms, we note that *PureRand* performs the poorest. This shows that the benefit obtained by incorporating a learning mechanism into the hyper-heuristic approaches is more evident in this problem than in the space allocation problem above. In the absence of an intelligent mechanism to 'learn' to choose a 'good' heuristic at each decision point, the *PureRand* algorithm is bound to perform poorly. It can also be said that, overall, *TSRoulWeel* and *MTSRoulWeel* seem to produce the best sets of non-dominated solutions. It is striking to note that *MTSRoulWeel* does not always get so close to the desired trade-off front. This (relatively) poor performance from *MTSRoulWeel* can be explained by the use of multiple tabu lists.

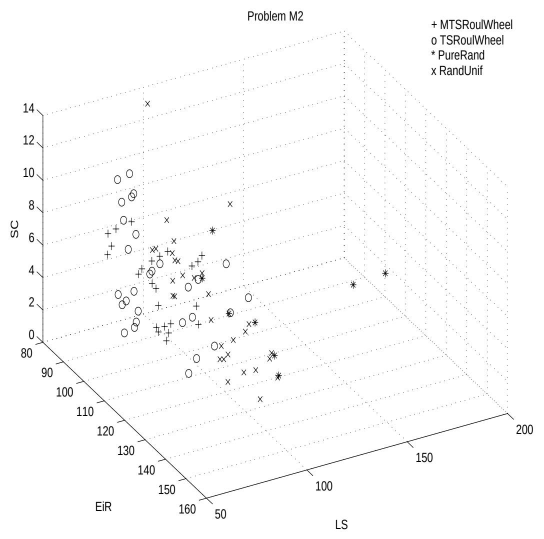


Figure 1.4. Non-dominated sets obtained with each of the hyper-heuristics for the problem M2.

Indeed, the use of several tabu lists may lead to an overhead in terms of tabu list management. As it was also observed in the application to the space allocation problem, it seems that having just one tabu list is good enough to produce solutions of acceptable quality. Perhaps, more elaborate mechanisms of dealing with multiple tabu lists of neighbourhood search heuristics can help to obtain better results in the *MTSRoulWheel*. We are currently exploring this possibility.

Comparison with Previous Results. As indicated above, we are tackling the course timetabling problem in a multi-objective fashion. In order to find out if the solutions produced by our approaches are competitive with those obtained with other methods, we compare our results with those reported in (Socha et al. 2002) and in (Burke et al. 2003b). Two tailored algorithms were used in (Socha et al. 2002), a local search approach (LLS) and an ant algorithm (ANT). In (Burke et al. 2003b), the approach presented was a single-objective tabu search hyper-heuristic. In those three algorithms, the problem is tackled as a single objective problem, i.e. using the aggregated value Scv. Here, we tackle the three objectives (LS,EiR, and SC) independently. In addition to the medium size instances M2, M3, and M4 used in the experiments above, we also compare the algorithms in the small size instance S1. To allow a comparison, we computed the aggregated objective value (Scv) for the non-dominated solutions obtained by each hyper-heuristic

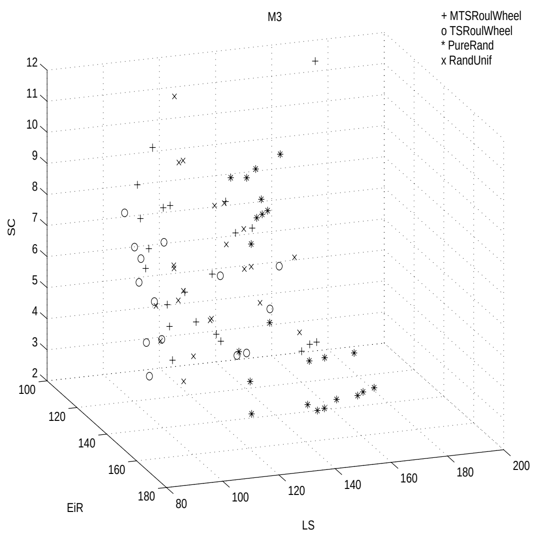


Figure 1.5. Non-dominated sets obtained with each of the hyper-heuristics for the problem M3.

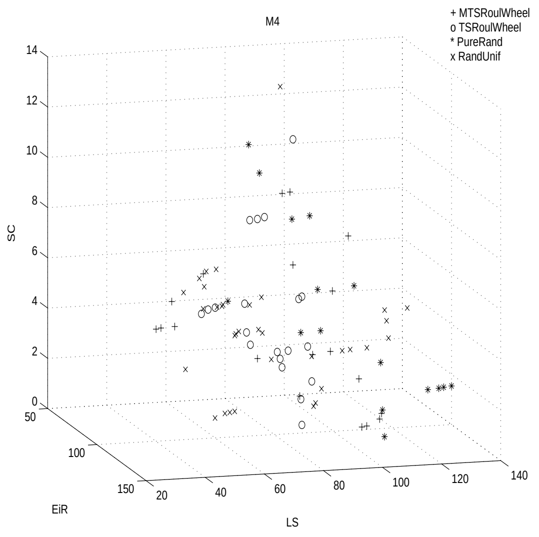


Figure 1.6. Non-dominated sets obtained with each of the hyper-heuristics for the problem M4.

	MTSRW	TSRW	TSRU	HH	LLS	ANT
S1	2.3/1	2.15/**0**	1.95/**0**	2.2/1	8	1
M2	198.4/**168**	196.5/173	219.2/192	197.6/173	202.5	184
M3	272.8/231	276.2/**224**	274.2/244	295.4/267	77.5% Inf	248
M4	189.2/**134**	191.8/160	190/149	180/169	177.5	164.5

Table 1.1. Comparison between the multi-objective tabu search hyper-heuristics with the two single-objective local search (LLS) and ant algorithms (ANT) from (Socha et al. 2002) and the single-objective hyper-heuristic (HH) approach from (Burke et al. 2003b). MTSRW is *MTSRoulWheel*, TSRW is *TSRoulWheel*, and TSRU is *TSRandUnif*. For all the hyper-heuristics the table shows average *Scv* (if solution is feasible) / best *Scv* in all runs. In column LLS 77.5% Inf indicates the proportion of infeasible solutions in 40 runs. The best obtained solutions are shown in bold.

approach. The best and average values for each off-line non-dominated set together with the results obtained in (Socha et al. 2002) and in (Burke et al. 2003b) are reported in Table 1.1. These results show that the sort of solutions produced by our multi-objective tabu search hyper-heuristics are of comparable quality with those reported in (Socha et al. 2002) and in (Burke et al. 2003b). The average solutions obtained by the multi-objective hyper-heuristics of this paper, are competitive with those obtained by the HH, LLS, and ANT approaches. Furthermore, for all instances the best solution is obtained by one of our multi-objective hyper-heuristic approaches. It should be noted that our algorithms used substantially fewer evaluations than the algorithms of (Socha et al. 2002). Overall, it can be said that our multi-objective hyper-heuristic approach is effective in tackling the three-objective course timetabling problem considered here.

6. The Conclusions

The problem of obtaining a uniformly distributed set of non-dominated solutions is of great concern in Pareto optimisation. This work proposes the application of hyper-heuristics for achieving a good coverage of the trade-off surface. The central idea is to develop a strategy that selects the most promising neighbourhood search heuristic in order to guide the search towards the desired areas of the trade-off surface. This technique has the advantage that it can be applied to single-solution and to population based algorithms because no population statistics are required as would be the case in, say, some clustering techniques. Experiments have been carried out on the space allocation problem and the university course timetabling problem. By using a hyper-heuristic approach for multi-objective combinatorial optimisation, the idea is to adapt the application of neighbourhood search heuristics according to the quality of the current solution in each of the objectives. In a way, this is similar to the strategy of tuning weights to specify search directions. The results obtained show that the hyper-heuristic approaches used here are capable of obtaining non-dominated sets that represent a good coverage of the trade-off surface. The results obtained in our experiments show that among the four approaches implemented here, the *TSRoulWheel* algorithm shows the best overall performance. The learning mechanism of the *TSRoulWheel* approach uses two strategies. One is Roulette-Wheel selection for deciding which objective to be tackled at any one time during the search. The *TSRoulWheel* algorithm also employs a single tabu list to manage the tabu status of the neighbourhood exploration heuristics during the search. Future work will concentrate on the improvement of the learning mechanism and upon further testing by comparing our approach with other multi-objective optimisers from the literature.

References

Ayob M., Kendall G. (2004). A Monte Carlo Hyper-Heuristic To Optimise Component Placement Sequencing for Multi Head Placement Machine. Proceedings of the 2003 International Conference on Intelligent Technologies (InTech 2003), pp. 132-141, Chiang Mai Thailand.

Burke E.K., Jackson K., Kingston J.H., Weare R. (1997). Automated University Timetabling: the State of the Art. The Computer Journal, Vol. 40, No. 9, pp. 565-571.

Burke E.K., Kendall G., Newall J., Hart E., Ross P., Schulemburg S. (2003). Hyper-heuristics: an Emerging Direction in Modern Search Technology. In: Glover F.W., Kochenberger G.A. (eds.), Handbook of Metaheuristics, Kluwer Academic Publishers.

Burke E.K., Kendall G., Soubeiga E. (2003b). A Tabu-search Hyper-heuristic for Timetabling and Rostering. Journal of Heuristics, Vol. 9, pp. 451-470.

Burke E.K., Landa Silva J.D. (2002). Improving the Performance of Multiobjective Optimisers by Using Relaxed Dominance. Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL 2002), Singapore, pp. 203-207.

Burke E.K., Landa Silva J.D. (2005). The Influence of the Fitness Evaluation Method on the Performance of Multiobjective Optimisers. To appear in European Journal of Operational Research.

Burke E.K., Newall J. (2004). Solving Examination Timetabling Problems Through Adaptation of Heuristic Orderings. Annals of operations Research, Vol. 129, pp. 107-134.

Burke E.K., Varley D.B. (1998). Space Allocation: An Analysis of Higher Education Requirements. The Practice and Theory of Automated Timetabling II: Selected Papers from the 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT 97), Lecture Notes in Computer Science, Springer, Vol. 1408, pp. 20-33.

Carter M.W., Laporte G. (1998). Recent Developments in Practical Course Timetabling. The Practice and Theory of Automated Timetabling

II: Selected Papers from the 2nd International Conference on the Practice and Theory of Automated Timetabling (PATAT 97), Lecture Notes in Computer Science, Vol. 1408, Springer, pp. 3-19.

Coello Coello C.A., Van Veldhuizen D.A., Lamont G.B. (2002). Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic Publishers.

Cowling P., Kendall G., Han L. (2002). An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), pp. 1185-1190.

Cowling P., Kendall G., Soubeiga E. (2000). A Hyperheuristic Approach to Scheduling a Sales Summit. The Practice and Theory of Automated Timetabling III: Selected Papers from the 3rd International Conference on the Practice and Theory of Automated Timetabling PATAT 2000, Lecture Notes in Computer Science, Vol. 2079, Springer, pp. 176-190.

Cowling P., Kendall G., Soubeiga E. (2001). Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. Second European Conference on Evolutionary Computing for Combinatorial Optimisation (EvoCop 2002), Lecture Notes in Computer Science, Vol. 2037, Springer, pp. 1-10.

Cowling P., Kendall G., Soubeiga E. (2002b). Hyperheuristics: A Robust Optimisation Method Applied to Nurse Scheduling. Proceedings of the VII Parallel Problem Solving From Nature (PPSN VII), Lecture Notes in Computer Science, Vol. 2439, Springer, pp. 7-11.

Czyzak P., Jaszkiewicz A. (1998) Pareto Simulated Annealing - A Metaheuristic for Multiple-objective Combinatorial Optimization. Journal of Multicriteria Decision Analysis, Vol. 7, No. 1, pp. 34-47.

Deb K. (2001). Multi-Objective Optimization Using Evolutionary Algorithms, Wiley.

Deb K., Manikanth M., Mishra S. (2003). Towards a Quick Computation of Well-Spread Pareto Optimal Solutions. Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Faro Portugal, Lecture Notes in Computer Science, Vol. 2632, Springer, pp. 222-236.

Farina M., Amato P. (2003). Fuzzy Optimality and Evolutionary Multiobjective Optimization. Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Faro Portugal, Lecture Notes in Computer Science, Vol. 2632, Springer, pp. 58-72.

Gaw A., Rattadilok P., Kwan R.S.K. (2004). Distributed Choice Function Hyper-Heuristics for Timetabling and Scheduling. Proceedings of the 2004 International Conference on the Practice and Theory of Automated Timetabling (PATAT 2004), Pittsburgh USA, pp. 495-497.

Glover F., Laguna M. (1997). Tabu Search. Kluwer Academic Publishers.

Goldberg, D. (1989). Genetic Algorithms in Search, Optimisation and Machine Learning. Addison Wesley.

Gunawan S., Farhang A., Azarm 5. (2003). Multi-level Multi-objective Genetic Algorithm Using Entropy to Preserve Diversity. Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Faro Portugal, Lecture Notes in Computer Science, Vol. 2632, Springer, pp. 148-161.

Han L., Kendall G. (2003). Investigation of a Tabu Assisted Hyper-Heuristic Genetic Algorithm. Proceedings of the 2003 Congress on Evolutionary Computation (CEC2003), Canberra Australia, pp. 2230-2237, IEEE Press.

Hansen P., Mladenovic N. (2001). Variable Neighbourhood Search: Principles and Applications. European Journal of Operational Research, Vol. 130, No. 3, pp. 449-467.

Hart E., Ross P. (1998). A Heuristic Combination Method for Solving Job-shop Scheduling Problems, Proceedings of the V Parallel Problem Solving From Nature (PPSN V), Lecture Notes in Computer Science, Vol. 1498, Springer, pp. 845-854.

Horn J. (2003). Niche Distributions on the Pareto Optimal Front, Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Faro Portugal, Lecture Notes in Computer Science, Vol. 2632, Springer, pp. 365-375.

Ishibuchi H., Yoshida T., Murata T. (2002). Selection of Initial Solutions for Local Search in Multiobjective Genetic Local Search. Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), Hawaii USA, pp. 950-955.

Ishibuchi H., Shibata Y. (2003). An Empirical Study on the Effect of Mating Restriction on the Search Ability of EMO Algorithms. Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Faro Portugal, Lecture Notes in Computer Science, Vol. 2632, Springer, pp. 433-447.

Jin H., Wong M.L. (2003). Adaptive Diversity Maintenance and Convergence Guarantee in Multiobjective Evolutionary Algorithms. Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003), Camberra Australia, IEEE Press, pp. 2498-2505.

Jones D.F., Mirrazavi S.K., Tamiz M. (2001). Multiobjective Metaheuristics: An Overview of the Current State-of-the-Art. European Journal of Operational Research, Vol. 137, No. 1, pp. 1-9.

Kaelbling L.P., Littman M.L., Moore A.W. (1996). Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research, Vol. 4, pp. 237-285.

Knowles J., Corne D.C. (2000). Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy, Evolutionary Computation, Vol. 8, No. 2, pp. 149-172.

Kokolo I., Hajime K., Shigenobu K., Failure of Pareto-based MOEAS, Does Non-dominated Really Mean Near to Optimal?, Proceedings of the 2001 Congress on Evolutionary Computation (CEC 2001), pp. 957-962, 2001.

Kumar R., Rockett P. (2002). Improved Sampling of the Pareto-front in Multiobjective Genetic Optimization by Steady-state Evolution: A Pareto Converging Genetic Algorithm, Evolutionary Computation, Vol. 10, No. 3, pp. 283-314.

Landa Silva J.D. (2003). Metaheuristic and Multiobjective Approaches for Space Allocation. PhD Thesis, School of Computer Science and Information Technology, University of Nottingham.

Laumams M., Thiele L., Deb K., Zitzler E. (2002). Combining Convergence and Diversity in Evolutionary Multiobjective Optimization. Evolutionary Computation, Vol. 10, No. 3, pp. 263-282.

Laumanns M., Zitzler E., Thiele L. (2001). On the Effects of Archiving, Elitism, and Density Based Selection in Multi-objective Optimization. Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001), Lecture Notes in Computer Science, Vol. 1993, Springer, pp. 181-196.

Lu H., Yen G.G. (2002). Rank-density Based Multiobjective Genetic Algorithm. Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), Hawaii USA, IEEE Press, PP. 944-949.

Martello S., Toth P. (1990). Knapsack Problems - Algorithms and Computer Implementations.Wiley.

Mostaghim S., Teich J. (2003). The Role of e-dominance in Multi-objective Particle Swarm Optimization Methods. Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003), Camberra Australia, IEEE Press, PP. 1764-1771.

Murata T., Ishibuchi H., Gen M. (2001). Specification of Genetic Search Directions in Cellular Multi-objective Genetic Algorithms. Proceedings of the 1st International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001), Lecture Notes in Computer Science, Vol. 1993, Springer, pp. 82-95.

Rosenthal R.E. (1985). Principles of Multiobjective Optimization. Decision Sciences, Vol. 16, pp. 133-152.

Ross P., Marin-Blazquez J.G., Schulenburg S., Hart E. (2003). Learning a Procedure that Can Solve Hard Bin-packing Problems: A New GA-based Approach to Hyper-heuristics. Proceedings of the 2003 Genetic and Evolutionary Computation Conference (GECCO 2003), Lecture Notes in Computer Science, Vol. 2724, Springer, pp. 1295-1306.

Ross P., Schulenburg S., Marin-Blazquez J.G., Hart E. (2002). Hyper-heuristics: Learning to Combine Simplke Heuristics in Bin-packing Problems. Proceedings of the 2002 Genetic and Evolutionary Computation Conference (GECCO 2002), Morgan Kaufmann, pp. 942-948.

Rossi-Doria O., Sampels M., Birattari M., Chiarandini M., Dorigo M., Gambardella L.M., Knowles J., Manfrin M., Mastrolilli M., Paechter B., Paquete L., Sttzle T. (2003). A Comparion of the Performance of Different Metaheuristics on the Timetabling Problem, The Practice and Theory of Automated Timetabling IV : Selected Papers from the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), Lecture Notes in Computer Science, Vol. 2740, Springer, pp. 330-352.

Salman F.S., Kalagnaman J.R., Murthy S., Davenport A. n(2002). Co-operative Strategies for Solving Bicriteria Sparse Multiple Knapsack Problem, Journal of Heuristics, Vol. 8, pp. 215-239.

Schaerf A. (1999). A Survey of Automated Timetabling. Artificial Intelligence Review, Vol. 13, pp. 87-127.

Socha K., Knowles J., Samples M. (2002). A Max-Min Ant System for the University Course Timetabling Problem. Ant Algorithms: Proceedings of the Third International Workshop (ANTS 2002), Lecture Notes in Computer Science, Vol. 2463, Springer, pp. 1-13.

Socha K., Kisiel-Dorohinicki M. (2002). Agent-based Evolutionary Multiobjective Optimization. Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), Hawaii USA, IEEE Press, PP. 109-114.

Soubeiga E. (2003). Development and Application of Hyperheuristics to Personnel Scheduling, PhD Thesis, School of Computer Science and Information Technology, University of Nottingham, June 2003.

Steuer Ralph E. (1986). Multiple Criteria Optimization: Theory, Computation and Application. Wiley.

Sutton R.S., Barto A.G. (1998). Reinforcement Learning, MIT Press.

Talbi E.G., Rahoudal M., Mabed M.H., Dhaenens C. (2001). A Hybrid Evolutionary Approach for Multicriteria Optimization Problems: Application to the Flow Shop. Proceedings of the 1st International Conference on Evolutionary Multi-criterion Optimization (EMO 2001), Lecture Notes in Computer Science, Vol. 1993, Zurich Switzerland, Springer, pp. 416-428.

Toscano Pulido G., Coello Coello C.A. (2003). The Micro Genetic Algorithm 2: Towards Online Adaptation in Evolutionary Multiobjective Optimization. Proceedings of the 2nd International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Lecture Notes in Computer Science, Vol. 2632, Springer, pp. 252-266.

Ulungu E.L., Teghem J. Fortemps P.H., Tuyttens D. (1999). MOSA Method: A Tool for Solving Multiobjective Combinatorial Optimization Problems. Journal of Multicriteria Decision Analysis, Vol. 8, pp. 221-236.

Zhu Z.Y., Leung K.S. (2002). Asynchronous Self-adjustable Island Genetic Algorithm for Multi-objective Optimization Problems. Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002), Hawaii USA, IEEE Press, pp. 837-842.