Multi-Step Environment Learning Classifier Systems applied to Hyper-Heuristics

Javier G. Marín-Blázquez
Department of Information and Communications Engineering (DIIC)
Facultad de Informática, Campus de Espinardo
Universidad de Murcia, 30100 Murcia, Spain
jgmarin@um.es

Sonia Schulenburg
Centre of Intelligent Systems and their Applications (CISA)
Appleton Tower, Room 3.14, Crichton Street
Edinburgh University, Edinburgh, EH8 9LE, United Kingdom
sonia.schulenburg@ed.ac.uk

ABSTRACT

Heuristic Algorithms (HA) are very widely used to tackle practical problems in operations research. They are simple, easy to understand and inspire confidence. Many of these HAs are good for some problem instances while very poor for other cases. While Meta-Heuristics try to find which is the best heuristic and/or parameters to apply for a given problem instance Hyper-Heuristics (HH) try to combine several heuristics in the same solution searching process, switching among them whenever the circumstances vary. Besides, instead to solve a single problem instance it tries to find a general algorithm to apply to whole families of problems. HH use evolutionary methods to search for such a problem-solving algorithm and, once produced, to apply it to any new problem instance desired. Learning Classifier Systems (LCS), and in particular XCS, represents an elegant and simple way to try to fabricate such a composite algorithm. This represents a different kind of problem to those already studied by the LCS community. Previous work, using single step environments, already showed the usefulness of the approach. This paper goes further and studies the novel use of multi-step environments for HH and an alternate way to consider states to see if chains of actions can be learnt. A non-trivial, NP-hard family of problems, the Bin Packing one, is used as benchmark for the procedure. Results of the approach are very encouraging, showing outperformance over all HAs used individually and over previously reported work by the authors, including non-LCS (a GA based approach used for the same BP set of problems) and LCS (using single step environments).

Categories and subject descriptors: I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search—Soft Computing

General Terms: Algorithms

Keywords: Soft Computing, Learning Classifier Systems, Hyper-Heuristics, Heuristics, Evolutionary Computation.

1. INTRODUCTION

Despite the supposed familiarity with the concept of Darwininan evolution and the success it shows in nature, bio-inspired algorithms and, in general, stochastic algorithms, are seen with suspicion by the common user. The facts that randomness is involved in the process, that it is difficult to provide guarantees of the quality of the solution, that many algorithms also have a sizeable set of parameters that need to be set (a process that can require considerable skill and experience), etc. produce a deep mistrust outside researchers in the field. Even renowned academical texts [14] can be very cautious about them, and users may justifiably prefer to use simpler deterministic approaches even if those approaches generally produce poorer results.

Besides, from the commercial viewpoint, there is not enough interest in solving optimisation problems to optimality, or even close to optimality. Instead the interest seems to be in obtaining "good enough - soon enough - cheap enough" kind of solutions. It is for these reasons that Heuristic Algorithms (HA) are very widely used to tackle practical problems in operations research. They are also simple, easy to understand and inspire confidence. Besides, many optimisation problems have simple heuristic methods to solve them, with the additional benefit that these heuristics may have associated theoretical best and worst performance limits.

Of course, many of these HAs are good for some problem instances while very poor for other cases. Lately there has been a lot of research in Meta-Heuristics, that usually tries to automatically find the best heuristic algorithm, and its most appropriate parameters, to be applied for a particular case.

A slightly different approach is taken by Hyper-Heuristics. On one side it will apply several heuristics at different stages of the process, whenever the circumstances suggest a more appropriate method than the currently used. On the other side it aims to produce a novel problem-solving algorithm rather than to solve individual problems. The idea is to use the evolutionary methods to search for such a problem-solving algorithm and, once produced, to apply it to any new problem instance desired. The novel algorithm is to be constructed from simple heuristic ingredients and should, ideally, be fast and deterministic and with good worst-case performance across a range of problems.

XCS represents an elegant and simple way to try to fabricate such a composite algorithm, and the interest in this work lies in assessing competitiveness of its performance.

It has already been presented in [13, 16], that even when the model was trained using only a few problems, it generalised and also performed well on a large number of unseen problems. That was a useful step towards the concept of using EAs to generate strong solution processes, rather than merely using them to find good individual solutions. Later work [10, 12, 11], using alternate EAs (a genetic algorithm) and applied to a different kind of problem (timetabling) provided further evidence of the benefits of the hyper-heuristics approach.

In [13] an XCS was applied using exclusively single step environments, meaning that rewards were available only after each action had taken place. Here the approach is extended to multi-step environments. Besides, there are two ways to define a *step*. One is to consider a step as performing a single action (in this work an action will be the filling of a single bin). Another approach is to consider a step as a sequence of actions (usually the same) until a change in the state of the environment is produced. Note that a state can have a large number of actions, and chains of various lengths can be selected for multi-step problems. This affects how and when rewards are applied. These approaches are tested using a large set of benchmark BP problems and a small set of eight heuristics, consisting of widely used algorithms, and combinations of these devised by the authors.

The work reported here attempts to find an algorithm that iteratively builds a solution, however it is important to note that this is atypical in the field. It is more common for an algorithm to start with a complete candidate solution and then search for improvements by making modifications, e.g. by some kind of controlled neighbourhood search. Hyper-heuristics could also be applied to try to discover a good, fast algorithm of that type, with tightly-bounded performance costs; this might be a topic for further research but outside the scope of the present work.

The rest of this paper is organised as follows: in section 2 the idea of hyper-heuristic is further developed and justified. Section 3 introduces one-dimensional bin-packing problems and some of their features. In section 4 the different heuristics used to solve the bin-packing problem are presented, followed by descriptions of how the state of the problem solving procedure is represented. Both, heuristics and state correspond, respectively, to the actions and the messages (context) for the XCS. Section 5 reports the experiments performed, including set-up and results obtained. The paper ends in section 6 were final conclusions are drawn.

2. THE IDEA OF HYPER-HEURISTICS

Despite the aforementioned mistrust from general practitioners with regards to evolutionary techniques, their use is often justified simply by results. Evolutionary algorithms can be excellent for searching very large spaces, at least when there is some reason to believe that there are 'building blocks' to be found. A 'building block' is a fragment, in the chosen representation, such that chromosomes which contain it tend to have higher fitness than those which do not. EAs bring building blocks together by chance recombination, and building blocks which are not present in the population at all may still be generated by mutation. Some EAs allow, as well, reinforcement learning, to be applied when some measure of reward can be granted to actions that produced good results. This type of learning procedure is specially useful in tasks where the solution to be found is a sequence of actions. In particular, learning classifier systems have repeatedly shown capabilities to deal with this kind of problems.

Hyper-Heuristics represents a step towards using EAs in a way that may solve some problems of acceptability for potential real-world use. The basic idea of HH is as follows: instead of using an EA to discover *a solution to a specific problem*, the EA is used to try to fabricate *a solution process applicable to many problem instances* built from simple, well-understood heuristics. Such a solution process might consist of using a certain heuristic initially, but after a while the nature of the remainder of the task may be such that a different heuristic becomes more appropriate to use. Once such solution process is discovered, it can be provided to users that can apply it as many times, and to as many different problems, as desired. The problems will be solved using the reassuring and simple heuristics that users are familiar with. A good general overview of hyper-heuristics and their applications can be found in [3].

The key idea in hyper-heuristics is to use members of a set of known and reasonably understood heuristics to transform the state of a problem. The key observation is a simple one: the strength of a heuristic often lies in its ability to make some good decisions on the route to fabricating an excellent solution. Why not, therefore, try to associate each heuristic with the problem conditions under which it flourishes and hence apply different heuristics to different parts or phases of the solution process?

The alert reader will immediately notice an objection to this whole idea. Good decisions are not necessarily easily recognisable in isolation. It is a sequence of decisions that builds a solution, and so there can be considerable epistasis involved - that is, a non-linear interdependence between the parts. However, many general search procedures such as evolutionary algorithms and, in particular, classifier systems, can cope with a considerable degree of epistasis, so the objection is not necessarily fatal.

Therefore, a possible framework for a hyper-heuristic algorithm:

1. Start with a set H of heuristic ingredients, each of which is applicable to a problem state and transform it to a new problem state.
2. Let the initial problem state be S_0
3. If the problem state is S_i then find the ingredient that is in some sense most suitable for transforming that state. Apply it, to get a new state of the problem S_{i+1}
4. If the problem is solved, stop. Otherwise go back to 3.

3. ONE-DIMENSIONAL BIN-PACKING PROBLEMS

In the one-dimensional Bin Packing problem (1DBPP), there is an unlimited supply of bins, each with capacity c (a positive number). A set of n items is to be packed into the bins, the size of item i is $s_i > 0$, and items must not over-fill any bin:

$$\sum_{i \in \text{bin}(k)} s_i \leq c \tag{1}$$

The task is to minimise the total number of bins used. Despite its simplicity, this is an NP-hard problem. If M is the minimal number of bins needed, then clearly:

$$M \geq \lceil (\sum_{i=1}^{n} s_i)/c \rceil \qquad (2)$$

and for any algorithm that does not start new bins unnecessarily, $M \leq$ bins used $< 2M$ (because if it used $2M$ or more bins there would be two bins whose combined contents were no more than c, and they could be combined into one).

A deeper discussion about algorithms used in literature to solve BP, including EAs, and the appropriateness of this benchmark for HH, can be found in [13, 16, 10].

3.1 Bin-packing Benchmark Problems

The problems used in this work come from two sources. The first collection is available from Beasley's OR-Library [1], and contains problems of two kinds that were generated and largely studied by Falkenauer [8]. The first kind, 80 problems named `uN_M`, involve bins of capacity 150. N items are generated with sizes chosen randomly from the interval 20-100. For N in the set $(120, 250, 500, 1000)$ there are twenty problems, thus M ranges from 00 to 19. The second kind, 80 problems named `tN_M`, are the triplet problems mentioned earlier. The bins have capacity 1000. The number of items N is one of 60, 120, 249, 501 (all divisible by three), and as before there are twenty problems per value of N. Item sizes range from 250 to 499 but are not random; the problem generation process was described earlier.

The second class of problems studied in this paper comes from the Operational Research Library [2] at the *Technische Universität Darmstadt*. Their 'bpp1-1' set and their very hard 'bpp1-3' set were used in this paper. In the bpp1-1 set problems are named `NxCyWz_a` where `x` is 1 (50 items), 2 (100 items), 3 (200 items) or 4 (500 items); `y` is 1 (capacity 100), 2 (capacity 120) or 3 (capacity 150); `z` is 1 (sizes in [1..100]), 2 (sizes in [20..100]) or 4 (sizes in [30..100]); and `a` is a letter in [A..T] indexing the twenty problems per parameter set. (Martello and Toth [9] also used a set with sizes drawn from [50..100], but these are far too easy to solve so they have been excluded from this work.) Of these 720 problems, the optimal solution is known in 704 cases and in the other sixteen, the optimal solution is known to lie in some interval of size 2 or 3. In the hard bpp1-3 set there are just ten problems, each with 200 items and bin capacity 100,000; item sizes are drawn from the range $[20,000..35,000]$. The optimal solution is known in only three cases, in the other seven the optimal solution lies in an interval of size 2 or 3. These results were obtained with an exact procedure called BISON [15] that employs a combination of tabu search and modified branch-and-bound.

In all, therefore, 890 benchmark problems are used.

4. COMBINING HEURISTICS WITH XCS

This section is divided into three parts. The first subsection describes the heuristics used, addressing why they were selected. The next subsection describes the representation used within XCS. Finally, how XCS is used to discover a good set of rules is explained.

4.1 The Set of Heuristics

First, a variety of heuristics were developed and their performances were evaluated on the benchmark collection. Of the fourteen that were implemented and tested, some were taken directly from the literature, others were variants created by the authors. Some of these algorithms were always dominated by others; among those that sometimes obtained the best of the fourteen results on a problem, some were always first-equal, rather than being *uniquely* the best of the set. There is no space here to describe the full set, but four, whose performance seemed collectively to be representative of the best, were selected as follows:

- Largest Fit Decreasing (LFD): items are taken in order of size, largest first, and put in the first bin where they will fit (a new bin is opened if necessary, and effectively all bins stay open). This was the best of the fourteen heuristics in over 81% (compared with other heuristics tested and not with the results reported in literature) of the bpp1-1 problems, but was never the winner in the bpp1-3 problems.
- Next-Fit-Decreasing (NFD): an item is placed in the current bin if possible, or else a new bin is opened, becoming the current bin where the item is placed. This heuristic usually performs very poorly.
- Djang and Finch's algorithm (DJD), see [7]. This puts items into a bin, taking items largest-first, until that bin is at least one third full. It then tries to find one, or two, or three items that completely fill the bin. If there is no such combination it tries again, but looking instead for a combination that fills the bin to within 1 of its capacity. If that fails, it tries to find such a combination that fills the bin to within 2 of its capacity; and so on. This of course gets excellent results on, for example, Falkenauer's problems; it was the best performer on just over 79% (again against other tested Heuristics and not with the results reported in literature) of those problems but was never the winner on the hard bpp1-3 problems.
- DJT (Djang and Finch, more tuples): devised by the authors, this corresponds to a modified form of DJD, considering combinations of up to five items rather than three items. In the Falkenauer problems, DJT performs exactly like DJD, as one would expect; in the bpp1-1 problems it is a little better than DJD.

In addition to these four, a 'filler' process was also used (coupled with each algorithm), which tried to find items to pack in any open bins, rather than moving on to a new bin. This might, for example, make a difference in DJD if a bin could be better filled by using more than three items once the bin was one-third full. Thus, in all, eight heuristics were used. The action of the filler process is described later.

4.2 Representing Problem State for XCS

As explained above, the idea is to find a good set of rules each of which associates a heuristic with some description of the current state of the problem. To execute the rules, the initial state is used to select a heuristic and that heuristic is used to pack a bin. The rules are then consulted again to find a heuristic appropriate to the altered problem state, and the process repeats until all items have been packed.

The problem state is reduced to the following simple description. The number of items remaining to be packed are examined, and the percentage R of items in each of four ranges is calculated. These ranges are shown in table 1.

Table 1: Item size ranges

Huge:	items over 1/2 of bin capacity
Large:	items from 1/3 up to 1/2 of bin capacity
Medium:	items from 1/4 up to 1/3 of bin capacity
Small:	items up to 1/4 of bin capacity

These are, in a sense, natural choices, since at most one huge item will fit in a bin, at most two large items will fit a bin, and so on. The percentage of items that lie within any one of these ranges is encoded using two bits as shown in table 2.

Table 2: Representing the proportion of items in a given range

Bits	Proportion of items
0 0	0 – 10%
0 1	10 – 20%
1 0	20 – 50%
1 1	50 –100%

Thus, there are two bits for each of the four ranges. Finally, it seemed important to also represent how far the process had got in packing items. For example, if there are very few items left to pack, there will probably be no huge items left. Thus, three bits are used to encode the percentage of the original number of items that still remain to be packed; table 3 illustrates this.

Table 3: Percentage of Items Left

Bits	% left to pack	Bits	% left to pack
0 0 0	0 – 12.5	1 0 0	50 – 62.5
0 0 1	12.5 – 25	1 0 1	62.5 – 75
0 1 0	25 – 37.5	1 1 0	75 – 87.5
0 1 1	37.5 – 50	1 1 1	87.5 – 100

The action is an integer indicating the decision of which strategy to use at the current environmental condition, as shown in table 4. As mentioned earlier, the second four actions use a filler process too, which tries to fill any open bins as much as possible. If the filling action successfully inserts at least one item, the filling step finishes. If no insertion was possible, then the associated heuristic (for example, LFD in 'Filler+LFD') is used. This forces a change in the problem state. It is important to remember that the trained XCS chooses deterministically, so that it is important for the problem state (if not the state description) to change each time, to prevent endless looping.

The alert reader might wonder whether the above problem state description in some way made heuristic selection an easy task. However, when each of our 14 original heuristics were evaluated, it was found that many cases where two problems had the same initial state description, but different algorithms, were the winners of the 14-way contest. For each of the 14 algorithms it was tried using a perceptron to see whether it was possible to classify problems into those on which a given algorithm was a winner and those on which it was not a winner. In every case, it was not possible, and therefore the learning task faced by XCS was not a trivial one.

Table 4: The action representation

Action	Meaning, Use	Action	Meaning, Use
000	LFD	100	Filler + LFD
001	NFD	101	Filler + NFD
010	DJD	110	Filler + DJD
011	DJT	111	Filler + DJT

5. THE EXPERIMENTS

The core of the application is Martin Butz' version of XCS [4, 5, 6], freely available over the web from the IlliGAL site. The reward scheme, including *when* and *what* to reward, lies in a level above the general XCS implementation. Some modifications were made for multi-step environment, which will be explained below, but the core remained the same, in an effort of providing more evidence of the usefulness and generality capabilities of XCS. The XCS parameters used were exactly as in [17], unless otherwise stated.

5.1 Set-up

5.1.1 Training and testing sets:

Each set of bin-packing problems is divided into training and test sets. In each case, the training set contained 75% of the problems; every fourth problem was placed in the test set. Since the problems come in groups of twenty for each type, the different sorts of problem were well represented in both training and test sets. All types of problems were combined into one large set of 890 problems that divided in training and test sets in the same manner. The reports below focus only on results by this combined collection, in which the training set has 667 problems and the test set has 223 problems. The combined set provides a good test of whether the system can learn from a very varied collection of problems.

5.1.2 Computational effort:

The experiments proceeded as follows. A limit of L for training cycles for XCS was set, where the values tried were $L = 10000, 15000, 20000, 25000, 30000$. During the learning phase, XCS first randomly chooses a problem to work on from the training set. One step, whether explore or exploit, usually corresponds to filling one bin (see below more about steps). In an explore step the action is chosen randomly, in an exploit step it is chosen according to the maximum prediction appropriate to the current problem state description. This is repeated until all the items in the current problem have been packed. A new random problem is then chosen. A cycle will always be considered as packing a single bin (some 'steps' can pack several bins and consume several cycles). Clearly, a large problem such as one of the `u1000_M` will consume a great number of cycles.

5.1.3 Recording the results:

The best result obtained on each problem during this training phase is recorded. Remember, however, that training continues, so the rule set may change after such best result was found. In particular, the final rule set at the end of the training phase might not be able to reproduce the best result ever recorded on every problem. Note as well that, during training, there is some exploration that uses random choices. Nevertheless, it is reasonable to record the best result ever found during (rather than at the end of) training on each problem also, because these are still reproducible results, by re-running the training with the same seed, and easily so. That best result obtained during training will be termed as TRB. It represents a potential achievable result for the hyper-heuristics.

At the end of training, the final rule set is used on every problem in the same training set to assess how well this rule set works. This is done using exploitation all the time so the result may be worse than the aforementioned best result recorded during the training process. This *reproducible* training result will be referred to as TRR.

Of course, the classifier obtained is also applied to every problem in the test set to measure the generalisation capabilities of the process. These test results will be shown under the label TST.

In summary, there would be two values for the training problems set: best ever explored (TRB) and the best recalled (TRR). Likewise, there is one value for the testing problems set (TST). All results shown are averages over ten runs with different seeds, unless stated otherwise.

In previous work [13, 10, 16] comparisons where made not only against the best results reported in literature, but also with the best result obtained by any of the constituent heuristics used. This makes sense, as a fair measure of the synergistic effect of hyper-heuristics is to compare it with the best of all of its individual components for each problem. However, since results of these comparisons were excellent –always above 98% and for the extra 2%, only one extra bin was used–, demonstrating the convenience of using hyper-heuristics over single heuristics has been set aside in this work. So in what follows, only comparisons against *best results reported in literature* will be provided.

5.1.4 Rewards:

The reward earned is proportional to how efficiently a bin was packed. For example, if a bin is packed to 94% of its capacity, then the reward earned is 0.94. Remember that 'packing' here means continuing to the point where the heuristic would switch bins, rather than optimally packing. A reward of 1.0 is paid for packing the final bin. Otherwise, an algorithm which, say, placed the final item of size 1 in a final bin in order to complete the packing, would earn only 0.01. The filler is rewarded in a slightly different way; it is rewarded in proportion to how much it reduces the empty space in the open bins.

5.1.5 Random Algorithm for comparison:

The problems were also solved using an algorithm that chooses randomly a heuristic to be applied. This is done to double check that the algorithm is learning indeed. The results are also averaged over 10 seeds as well. However, the random choice of the heuristic solved optimally only 50.4% and 54.9% of the problems on the training and test sets respectively, and the HH reaches over 80%. Since the value is so low, it is not included in the following figures.

5.2 Alternatives explored

This paper will focus in comparing a single step environment versus a multi-step environment using a step by *state*. A deeper study, with more alternatives and learning parameters is currently under revision.

5.2.1 Step by State/Action:

In [13] one step is defined as packing one bin (LFD was modified to pack only one bin and stop). From deeper study of the sequences of actions applied and the status of the problems, it was clear that in many cases the filling of a single bin does not change the status itself, given the binary representation proposed. It is easy to see that, as what is represented is ranges, and not absolute values, slight changes may not be reflected immediately in a change of status. As an alternative, one step can be considered to be taking a 'single action' (the repetition of the same single bin packing action) until there is a change of state.

The reward is the average of the different rewards obtained in each action of packing a single bin, and is only applied once at the change of state. For example, if four actions of type 2 were applied to perform a single step by state −2222− and each action has rewards of, say: 990, 560, 230, and 870, the reward given to the action set (the rules that suggested the action) would be 990+560+230+870 = 2650/4 = 662.5, and it will be rewarded only once.

5.2.2 Single Step and Multiple Step Environment:

As mentioned, in single step environments, rewards are available at every step (being this step delimited by either a change of state or action). A multi-step environment allows to consider sequences of steps and be rewarded together as a chain. This means that the reward can be paid after performing a number of complete states, or individual actions (depending if step by state is used or not, respectively). In these cases it is likely that several and different actions are applied in one multi-step trial. Several multi-step chain sizes have been tried, namely chains of size 2, 5 and 10 actions.

5.2.3 Rewarding Single Step and Multiple Step (with Step by State)

Given a hypothetical sequence of actions (dash represent a change in state) as follows:

$$-\underbrace{\overset{r_1}{\overset{\uparrow}{2}}\,\overset{r_2}{\overset{\uparrow}{2}}\,\overset{r_3}{\overset{\uparrow}{2}}}_{S_1}-\underbrace{\overset{r_4}{\overset{\uparrow}{1}}\,\overset{r_5}{\overset{\uparrow}{1}}}_{S_2}-\underbrace{\overset{r_6}{\overset{\uparrow}{3}}\,\overset{r_7}{\overset{\uparrow}{3}}\,\overset{r_8}{\overset{\uparrow}{3}}\,\overset{r_9}{\overset{\uparrow}{3}}}_{S_3}-\underbrace{\overset{r_{10}}{\overset{\uparrow}{2}}}_{S_4}-\underbrace{\overset{r_{11}}{\overset{\uparrow}{1}}\,\overset{r_{12}}{\overset{\uparrow}{1}}\,\overset{r_{13}}{\overset{\uparrow}{1}}}_{S_5}-\underbrace{\overset{r_{14}}{\overset{\uparrow}{1}}}_{S_6}-$$

Here, there are 6 different states and 14 different actions of 3 types. Each of the 14 actions produces a potential reward r_i;$1 \leq i \leq 14$ based, as mentioned, in how efficient was the packing of a single bin.

In the sequence below, underlines will represent where the reward pointed by the upwards arrow is applied. Overbraces will represent how the rewards to be used are obtained (by averaging).

If Single Step and Step by State is used, then the rewards are averaged for each state, and rewards applied only once per state, therefore rewards were given 6 times, as follows:

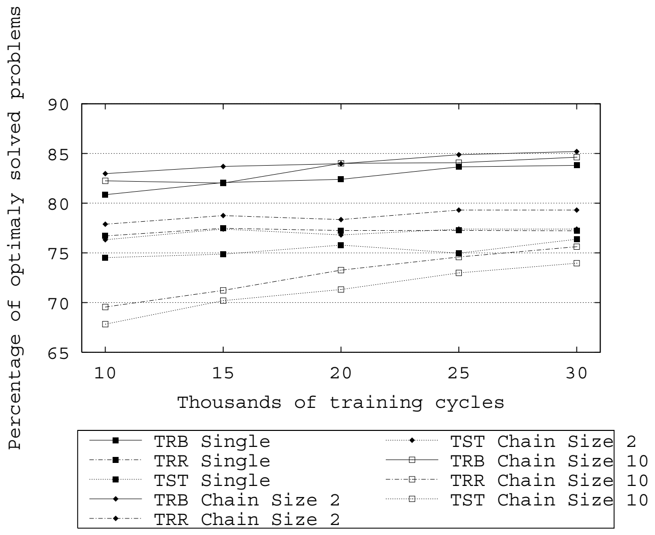


Figure 1: Computational Effort

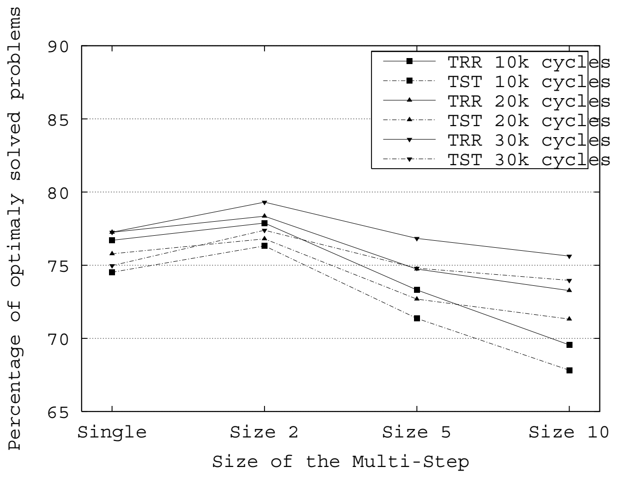


Figure 2: Single vs Multiple step approach

$$\overbrace{(\underline{222})}^{R_1}\ \overbrace{(\underline{11})}^{R_2}\ \overbrace{(\underline{3333})}^{R_3}\ \overbrace{(\underline{2})}^{R_4}\ \overbrace{(\underline{111})}^{R_5}\ \overbrace{(\underline{1})}^{R_6}$$
$$\underset{R_1}{\uparrow}\quad \underset{R_2}{\uparrow}\quad \underset{R_3}{\uparrow}\quad \underset{R_4}{\uparrow}\quad \underset{R_5}{\uparrow}\quad \underset{R_6}{\uparrow}$$

$$\begin{array}{ll} R_1 = \frac{r_1+r_2+r_3}{3} & R_2 = \frac{r_4+r_5}{2} \\ R_3 = \frac{r_6+r_7+r_8+r_9}{4} & R_4 = r_{10} \\ R_5 = \frac{r_{11}+r_{12}+r_{13}}{3} & R_6 = r_{14} \end{array} \tag{3}$$

On the other hand, if Multi-Step (size 2) is used, then rewards are calculated as the averages of the rewards of the states. As the step here is performed by state, these rewards by state (M_s) are obtained by averaging the individual rewards of the different actions that were executed in that step. The obtained rewards (R_m) are awarded only once per state. Therefore there are 3 different reward values, and the rewarding procedure is performed 6 times:

$$\overbrace{\overbrace{M_1}\ \overbrace{M_2}}^{R_1}\ \overbrace{\overbrace{M_3}\ \overbrace{M_4}}^{R_2}\ \overbrace{\overbrace{M_5}\ \overbrace{M_6}}^{R_3}$$
$$(\underline{222})\ (\underline{11})\ (\underline{3333})\ (\underline{2})\ (\underline{111})\ (\underline{1})$$
$$\underset{R_1}{\uparrow}\quad \underset{R_1}{\uparrow}\quad \underset{R_2}{\uparrow}\quad \underset{R_2}{\uparrow}\quad \underset{R_3}{\uparrow}\quad \underset{R_3}{\uparrow}$$

$$\begin{array}{lll} M_1 = \frac{r_1+r_2+r_3}{3} & M_2 = \frac{r_4+r_5}{2} & M_3 = \frac{r_6+r_7+r_8+r_9}{4} \\ M_4 = r_{10} & M_5 = \frac{r_{11}+r_{12}+r_{13}}{3} & M_6 = r_{14} \\ R_1 = \frac{M_1+M_2}{2} & R_2 = \frac{M_3+M_4}{2} & R_3 = \frac{M_5+M_6}{2} \end{array} \tag{4}$$

5.3 Results

5.3.1 *Computational Effort:*

Figure 1 shows average runs for single step and multi-steps of size 2 and 10 drawn against the number of training cycles. As it can be expected, the more cycles allowed to the classifier, the better the results. Nevertheless, there are differences between the best ever obtained while training (TRB) and what can be later recalled (TRR) on the same training problems (see section 5.1.4). While the former is obtaining better and better results, the latter seems to level near the 80%. This points to the fact that it seems that although the heuristics can be combined to achieve the results given by TRB values, the classifier, in exploiting mode (TRR), is unable to reproduce them later. It seems that the representation of the state is too poor to provide enough information about situations where other actions are more fruitful. Remember that in exploiting mode the same action is performed until the state representation changes, as expected of a deterministic algorithm. There is sufficient evidence of the learning capabilities of XCS, even within the limitations of its binary representation, so research was conducted to evaluate the effects of using more information for the state representation [12, 11] and with alternative learning methods using real numbers [10]. There was a substantial increase in performance, reaching values close to the best of training. This seems to support the idea that the state representation used can be improved and that XCS will then be provided with enough information to properly discriminate. Experiments with extended status representations using XCS are proposed as further research.

Single step seems to have a ceiling of performance at around 78% on TRR values (on the averaged results). This ceiling is reached with just 10000 cycles. More cycles do not improve the results significantly. Figure 2 shows the effect on performance for several different training efforts with regards to the length of the chains used in single and multi-step environments. There, the small impact of increasing the number of cycles on single step environments can be seen. It seems to learn fast what it can learn. But that is below what multi-step environments can achieve, albeit with more computational effort, as will be seen next.

5.3.2 *Single vs Multiple step approach:*

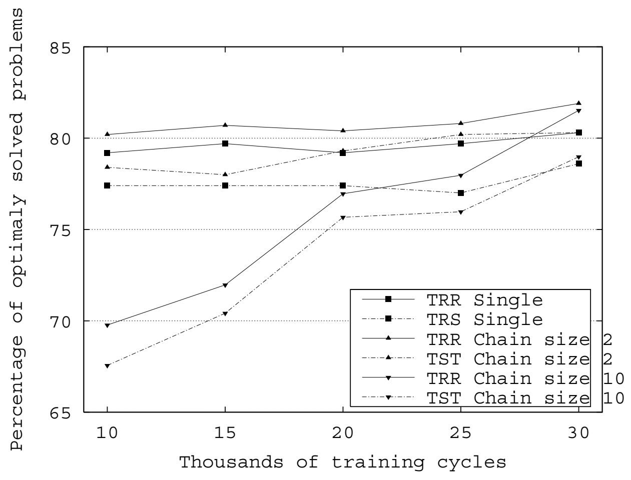
In [16] early experiments with multi-step (size 2) provided promising results on its impact in hyper-heuristic bin packing using XCS. When, in this work, larger chains were tested, a decrease in performance was clearly revealed. While size 2 outperformed single step, it failed to generalise with longer chains. It needs many more training cycles to achieve closer results. Figure 2 shows how the longer the chain, the more training needed.

This result of a performance peak on size 2 chains while it decreases on longer chains is very interesting. It suggests that there must be pairs of actions that seem to be useful enough, one after the other, as to easily emerge after the reinforcement learning. Unfortunately, longer chains seem not to be as useful for the bin packing problem and need many more cycles to achieve slightly better results.

Of course, it may also be that 10000, the minimum number of cycles chosen, is enough for the size 2 chains to show its potential and that single environments have reached their ceiling some cycles before. But this then highlights the potential of multi-step environments, as the ceilings seem to be above single step ones. The issue then will be at which computational cost to achieve a gain. Here 10000 cycles is enough for multi-step environments to start to show its potential.

5.3.3 *Best runs:*

While the results provided are averages, on several runs of the XCS training, to show the general behaviour expected, it is true that a single run is usually easily affordable (with respect to computer use) in applications that are not time-critical. This means that several runs with different seeds can be computed, and the best chosen as the final product. Figure 3 shows the best results (here only TRR and TST are shown) out of the 10 runs for single step and chain sizes 2 and 10, drawn against the number of cycles allowed for training. Here it is more clear that longer chain schemes may catch up with shorter ones, but at the cost of much more training. This graph also hints that with the configuration used, the status chosen and the heuristics available, the best recallable XCS is at around the 82% mark, with 80% on the testing set. The best attainable ever found (that is, the best TRB result) reached an exceptional 86.1% of optimality.



..

Figure 3: Best results of some experiments

5.4 Generalisation properties

Although not previously mentioned, it seems clear from the figures that the results on the testing set where similar to those obtained in the training set, a good indication that the system is able to generalise well. Results of the exploit steps during training are very close to results using a trained classifier on new test cases. This means that particular details learnt (a structure of some kind) during the adaptive phase (when the classifier rules are being modified according to experience, etc.) can be reproduced with completely new data (unseen problems taken from the test sets). As an example, as already reported in [13], for one of Falkenauer's problems, DJD (and our DJT) produced a new best, and optimal result (this had already been reported in [7] where DJD was described). Even if this problem is excluded from the training set, the learnt rule set can still solve it optimally.

6. CONCLUSIONS

This work has extended the experiments and applied some of the suggested enhancements of previous work [13, 10, 16] on applying hyper-heuristics to the bin packing problem using learning classifiers.

It will be appreciated that the scope of this work is broad. Attempts have been made to address several important issues, including narrowing the gap between the use of heuristic algorithms and evolutionary algorithms, and that of academic reports, and real-world applications. In summary, XCS was able to create a solution process that performed competitively well on a large collection of NP-hard, non-trivial problems found in literature. The system performed better than any of its constituent individual heuristics, and always performed better than the worst of the algorithms involved.

In particular, when using the heuristics independently, the best one of all (our own DJD version), achieved optimality in around 73% of the problems. However, the evolved XCS rule-sets were able to produce optimal solutions for over 78% of the problems (on unseen testing problems) in single step environments, and over 81% in multi-step environments (also on unseen testing problems), while in the rest it produced a solution very close to optimal. The best attainable found was 86.1%, which indeed represents an exceptionally good result given the ingredients used, and a considerable improvement from previous work reported.

Experiments showed that multistep environments of small size, under the right conditions outperform single step results. In addition, results seem to point out that multi-step environments can obtain better results than the single step environments could ever achieve. The ceiling, after which an increase of training cycles does not provide better results, appears to be higher in multi-step environments.

A longer version of this work, comparing the effects of alternate reward schemas, the use of step by state and by action, and other exploration/explotation policies during training is currently under revision.

Acknowledgements

Early developments of this work were funded by UK EPSRC research grant number GR/N36660 ending July 2004. The authors would like to thank Peter Ross and Emma Hart for their contributions during that period, and Stewart Wilson for useful comments and discussions.

Recent developments have been carried out by the authors sponsored by the Ramón y Cajal Research Program in the Intelligent Systems Research Group at the University of Murcia, Spain, and by Scottish Enterprise's Royal Society of Edinburgh Enterprise Fellowship, at the Centre of Intelligent Systems and their Applications, University of Edinburgh, Scotland.

7. REFERENCES

[1] http://www.ms.ic.ac.uk/info.html

[2] http://www.bwl.tu-darmstadt.de/bwl3/forsch/projekte/binpp/

[3] Edmund Burke, Emma Hart, Graham Kendall, Jim Newall, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An Emerging Direction in Modern Search Technology. In Fred Glover and Gary Kochenberger, editors, *Handbook of Meta-heuristics*, chapter 16, pages 457–474. Kluwer, 2002.

[4] Martin V. Butz. An Implementation of the XCS classifier system in C. Technical Report 99021, The Illinois Genetic Algorithms Laboratory, 1999.

[5] Martin V. Butz. XCSJava 1.0: An Implementation of the XCS classifier system in Java. Technical Report 2000027, Illinois Genetic Algorithms Laboratory, 2000.

[6] Martin V. Butz and Stewart W. Wilson. An Algorithmic Description of XCS. Technical Report 2000017, Illinois Genetic Algorithms Laboratory, 2000.

[7] Philipp A. Djang and Paul R. Finch. Solving One Dimensional Bin Packing Problems. *Journal of Heuristics*, 1998.

[8] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[9] Silvano Martello and Paolo Toth. *Knapsack Problems. Algorithms and Computer Implementations*. John Wiley & Sons, 1990.

[10] Peter Ross, Javier Marín-Blázquez, Sonia Schulenburg, and Emma Hart. Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics. In E. Cantú-Paz, J. A. Foster, K. Deb, D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. Dowsland, N. Jonoska, and J. Miller, editors, *Genetic and Evolutionary Computation Conference – GECCO 2003*, volume 2 of *Lecture Notes in Computer Sscience*, pages 1295–1306, Chicago, IL, USA, 2003. Springer.

[11] Peter Ross and Javier G. Marín-Blázquez. Constructive hyper-heuristics in class timetabling. In *CEC 2005*, pages 1493–1500, Edinburgh, Scotland, 2-5 September 2005. IEEE Press.

[12] Peter Ross, Javier G. Marín-Blázquez, and Emma Hart. Hyper-heuristics applied to class and exam timetabling problems. In *CEC 2004*, pages 1691–1698, Portland, Oregon, 20-23 June 2004. IEEE Press.

[13] Peter Ross, Sonia Schulenburg, Javier G. Marín-Blázquez, and Emma Hart. Hyper-heuristics: learning to combine simple heuristics in bin packing problems. In *Genetic and Evolutionary Computation Conference – GECCO 2002*, pages 942–948, New York, NY, USA, 2002. Morgan Kauffmann Publishers.

[14] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*, chapter 4: Informed Search and Exploration, pages 116–120. Prentice Hall, Upper Saddle River, N.J., 2nd international edition edition, 2003.

[15] Armin Scholl and Robert Klein. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research*, 1997.

[16] Sonia Schulenburg. A Hyper-Heuristic Approach to Single and Multiple Step Environments in Bin-Packing Problems. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson (workshop organisers), editors, *5th International Workshop on Learning Classifier Systems (IWLCS) 2002*. Unpublished contribution, presented, Granada, Spain, September 7-8, 2002.

[17] Stewart W. Wilson. Classifier Systems Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.