On the Life-Long Learning Capabilities of a NELLI*: A Hyper-Heuristic Optimisation System

Emma Hart and Kevin Sim

Institute for Informatics and Digital Innovation, Edinburgh Napier University
Merchiston Campus, Edinburgh, EH10 5DT, UK
{e.hart,k.sim}@napier.ac.uk

Abstract. Real-world applications of optimisation techniques place more importance on finding approaches that result in acceptable quality solutions in a short time-frame and can provide robust solutions, capable of being modified in response to changes in the environment than seeking elusive global optima. We demonstrate that a hyper-heuristic approach NELLI* that takes inspiration from artifical immune systems is capable of life-long learning in an environment where problems are presented in a continuous stream and change over time. Experiments using 1370 bin-packing problems show excellent performance on unseen problems and that the system maintains memory, enabling it to exploit previously learnt heuristics to solve new problems with similar characteristics to ones solved in the past.

Keywords: Hyper-heuristics, artificial immune systems.

1 Introduction

Hyper-heuristics cover a general class of search methods that attempt to automate the process of selecting, combining, generating or adapting simple heuristics in order to solve large classes of problems. Although some compromise in solution quality is likely when comparing the quality of any single solution to a specifically tuned optimisation algorithm, the motivation is that this is compensated for by guaranteeing acceptable performance across very large problem sets, using cheap heuristics that are often simple to understand and can incorporate human knowledge.

Online hyper-heuristic methods [4] typically learn a sequence of low-level heuristics that can be applied to perturb an existing solution and learn during the solving phase. In contrast, *offline* methods attempt to find mapping between problem state and heuristic in order to determine how to solve a problem, requiring an initial offline training period using a representative set of problems (e.g. [18]). Both approaches potentially suffer from weakenesses. In the former, the hyper-heuristic learns from scratch each time a new instance of a problem is solved. In the latter, if the characteristics of the problem set change overtime, the hyper-heuristic needs to be periodically retuned. This potentially leads to inefficient algorithms that both fail to exploit previously learned knowledge in the search for a solution and cannot adapt to changing characteristics of problems.

T. Bartz-Beielstein et al. (Eds.): PPSN XIII 2014, LNCS 8672, pp. 282–291, 2014.

Recently, Sim *et al* [13] proposed that the hyper-heuristic field could learn from new research in the machine-learning field in *moving beyond learning algorithms to more seriously consider the nature of systems that are capable of learning over a life-time*, stating that such systems must be capable of retaining knowledge (i.e. incorporate a memory) and of selectively applying that knowledge to new tasks. They described an approach inspired by Artificial Immune Systems (AIS) dubbed NELLI in which novel heuristics were continuously generated and a self-organising process determined whether they should be integrated into a self-sustaining network of problems and heuristics that could be used to solve a stream of problems continuously presented to the network. Results in [13] demonstrated that the system maintained memory, and was adaptable and efficient at solving problems when a dynamic stream of instances was presented. A modification to one of the key elements of the system — generating novel heuristics — was described in [15] (NELLI*) but was only demonstrated on static sets of problems in which the nature of the instances did not vary over time. Here, we demonstrate that using the new representation, not only is the system capable of life-long-learning but also that significant improvement is found over previous results when tested on a large corpus of bin-packing problems that vary in their characteristics and are presented in a continuous stream over time.

2 Background

In the hyper-heuristic domain, there are many examples of systems that either *generate* novel heuristics or *select* from a set of pre-defined heuristics to solve a problem and are shown to be capable of good performance across large problem sets, see [1] for a recent and comprehensive overview. However there is relatively little work that combines both generation and selection (recent examples include [7,9,17]). Additionally, although there is a wealth of work within the optimisation field addressing dynamic optimisation in which the fitness function applied to a single problem instance changes over time (e.g. [19]), we are not aware of other work that tackles problems in which the fitness function remains static but the characteristics of the instances presented varies over time.

In contrast, in the AIS literature, there are examples of systems that exhibit lifelong learning in response to changing environments, in particular in robotics. Typically, 'antibodies' specifying possible actions are connected in a plastic network that varies both in its topology and its constitution over time and determines an appropriate action to execute. This is typified by Whitbrook [20] who describes scenarios exhibiting both memory and adaptation in a robotics application — work which provided inspiration for NELLI*.

3 NELLI* Algorithm

Described in detail in [13] and visualised in Figure 1, the first version of NELLI comprised of three main parts: a stream of problem instances, a continuously generated stream of novel heuristics and a network that sustains co-stimulating components (heuristics and problem instances). NELLI is designed to run continuously; problem instances and heuristics can be added in any quantity at

any point. An AIS is responsible for governing the dynamic processes that enable heuristics and problem instances to be incorporated (stimulated) or rejected (suppressed) by the network. The orignal version ([13]) exploited a representation borrowed from Single Node Genetic Programming (SNGP) [6] in which a heuristic was represented by a tree randomly constructed from a set of terminal nodes that encapsulated information about the problem state (e.g. in the bin-packing domain, the free space in the bin and item size) and simple function nodes. All heuristics were randomly generated in this manner. In [15] two improvements were made. Firstly, the tree representation was replaced by a linear sequence of heuristic-components, each of which explicitly results in some items to be placed into the solution; secondly, rather than randomly generate all heuristics, a proportion p_m were generated by applying one of five mutation operators to an existing heuristic randomly chosen from the sustained network. By varying the value of p_m, it is possible to alter the balance between *exploration* (random generation of heuristics) and *exploitation* (focus the search around existing heuristics). The exploitation process is achieved through five operators:

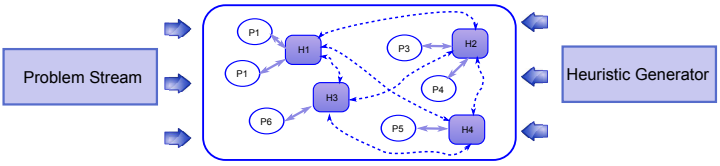


Fig. 1. A conceptual view of the system: Problems and heuristics are continuously injected into the AIS. The dynamics and meta-dynamics of the system result in a self-sustaining network of heuristics and problems. Heuristics and problems that receive no stimulation are removed.

- Select a random heuristic and swap the position of two random nodes.
- Select a random heuristic and replace a random node with a randomly selected node.
- Select a random heuristic from the network and remove a random node.
- Select a random heuristic and add a random node at a random position.
- Select two random heuristics from the network and concatenate their nodes.

Figure 2 shows a generic example of a heuristic represented by a string of five heuristic components with a "pointer" used by an encompassing wrapper to indicate the current component position. Each component is chosen from the list of nodes shown. If a node is successful in packing one or more items into a bin, then the pointer is advanced to the next node and the process continues with the current bin – when a node fails, a new bin is opened, and the pointer advances. The pointer is returned to the start after evaluation of the last node in the sequence. Each heuristic contains a sequence of nodes that are instantiated randomly up to a maximum *initial* length, specified by a parameter of the algorithm (l_{max}). The sequence may alter in length during the course of a run — the only constraint imposed is that the sequence must retain at least one node.

The network sustains a network of heuristics and problems through a process that varies the concentration of each element based on its *stimulation*. Problems

are *directly* stimulated by heuristics, and vice versa. Heuristics are *indirectly* stimulated by other heuristics. The total stimulation of a heuristic is the sum of its affinity with each problem in the set $\mathcal{P}$ currently in the network $\mathcal{N}$. A heuristic h has a non-zero affinity with a problem $p \in \mathcal{P}$ *if and only if* it provides a solution that uses fewer bins than any other heuristic currently in $\mathcal{H}$. If this is the case, then the value of the affinity $p \leftrightarrow h$ is equal to the improvement in the number of bins used by h compared to the next-best heuristic. If a heuristic provides the best solution for a problem p but one or more other heuristics give an equal result, then the affinity between problem p and the heuristic h is zero. If a heuristic h uses more bins than another heuristic on the problem, then the affinity between problem p and the heuristic h is also zero. This is shown mathematically in Equations 1 and 2 described in detail in [13]. Essentially, the equations favour *heuristics* that are able to find a niche in solving at least one problem better than any other heuristic in the system, and *problems* that represent niche regions of the instance space, i.e. two or more heuristics do not perform equally on them.

$$h_{stim} = \sum_{p \in \mathcal{P}} \delta_{bins} \begin{cases} \delta_{bins} = \min\left(bins_{\mathcal{H}'_p}\right) - bins_{h_p} & : \text{ if } \min\left(bins_{\mathcal{H}'_p}\right) - bins_{h_p} > 0 \\ \delta_{bins} = 0 & : \text{ otherwise} \end{cases} \tag{1}$$

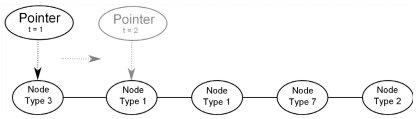
$$p_{stim} = \sum_{h \in \mathcal{H}} \delta_{bins} \begin{cases} \delta_{bins} = \min\left(bins_{\mathcal{H}'_p)}\right) - bins_{h_p} & : \text{ if } \min\left(bins_{\mathcal{H}'_p}\right) - bins_{h_p} > 0 \\ \delta_{bins} = 0 & : \text{ otherwise} \end{cases} \tag{2}$$

Algorithm 1. NELLI* Pseudo Code

Require: $\mathcal{H} = \emptyset$:The set of heuristics
Require: $\mathcal{P} = \emptyset$:The set of current problems
Require: $\mathcal{E} = \mathcal{E}_{t=0}$:The set of problems to be solved at time t
repeat
  optionally replace $\mathcal{E}$: $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup \mathcal{E}$
  repeat
    With probability p_m generate a new heuristic via mutation
    With probabilty $1 - p_m$ generate a new heuristic via random initialisation
  until n_h new heuristics generated
  Add n_h new heuristics to $\mathcal{H}$ with concentration c_{init}
  Add n_p randomly selected problem instances from $\mathcal{E}$ to $\mathcal{P}$ with concentration c_{init}
  calculate $h_{stim} \forall h \in \mathcal{H}$ using Equation 1
  calculate $p_{stim} \forall p \in \mathcal{P}$ using Equation 2
  increment all concentrations (both $\mathcal{H}$ and $\mathcal{P}$) that have concentration $< c_{max}$ and stimulation > 0 by Δ_c
  decrement all concentrations (both $\mathcal{H}$ and $\mathcal{P}$) with stimulation ≤ 0 by Δ_c
  Remove heuristics and problems with *concentration* ≤ 0
until *stopping criteria met*

4 NELLI* as a Life-Long-Learning System

In [15] we demonstrated that the linear representation described above improved the performance of NELLI in terms of finding optimal solutions when applied to a large but static set of problems. However, the capabilities of NELLI* as a life-long learning system were not demonstrated. The goal of the paper is to address this, showing that NELLI* is able to



Node Type	Description
1	Packs the single largest item into the current bin
2	Packs the largest combination of exactly 2 items into the current bin
3	Works as for 1 but packs exactly 3 items
4	Works as for 2 but packs exactly 4 items
5	Works as for 2 but packs exactly 5 items
6	Packs the largest combination of up to 2 items into the current bin giving preference to sets of lower cardinality.
7	As for 5 but considers sets of up to 3 items
8	As for 5 but considers sets of up to 4 items
9	As for 5 but considers sets of up to 5 items

Fig. 2. Heuristics are represented as linear sequences of nodes. A pointer keeps track of which node to apply next. The sequence restarts from the beginning after the last node is processed.

1. Demonstrate memory by quickly (re)finding solutions to problems that were seen by the system in the past
2. Continue to learn by continuing to improve its performance on problems in the datasets
3. Generalise, by quickly finding good solutions to problems that are similar to instances seen previously

We demonstrate this using a set of 1370 bin-packing problems taken from a variety of sources in the literature. Data sets $ds1, ds2$ & $ds3$ were introduced by [11] and comprise 720, 480 and 10 of the instances respectively. Problems in $ds1$, $ds3$ have optimal solutions with on average 3 items per bin and are similar in nature; solutions for problems in $ds2$ which have widely variable item weights have between 3 and 9 items per bin. Literature indicates that for a given algorithm, performance varies greatly on $ds2$ when compared to $(ds1, ds3)$ [5]. The remaining instances are taken from $FalU$, $FalT$, and were introduced by [3]. All 1370 problems have known optimal solutions. In the following discussion we distinguish the following :

- $\mathcal{U}$ - the set of 1370 problems from the class of 1D-BPP of interest.
- $\mathcal{E}$ - the current environment, i.e. the set of problems we are currently interested in solving, $\mathcal{E} \subset \mathcal{U}$
- $\mathcal{E}^*$ - the set of problems presented to the network so far
- $\mathcal{P}$ - the set of problems currently sustained in the immune network $\mathcal{P} \subset \mathcal{E}^*$ (this is an internal property of the network)

5 Results

Four experiments were conducted using parameters described in Table 1 (taken from [15] where the tuning process is described). Performance was evaluated in terms of the number of problems solved optimally and in terms of the number of bins required over the known optimal solutions.

Clearly the problems considered have been solved by a plethora of optimisation methods in the past (including hyper-heuristics). For instance, Burke *et al* [2] obtain excellent results on the 90 problem instances in $FalU$ and $ds3$ by evolving an individual heuristic per problem instance using 50000 evaluations for each instance. Others seek optimality using exact methods (e.g. [11]) to solve each instance separately. In contrast, NELLI aims to find high quality solutions to very large sets of problems by only evaluating a very small subset of the instance space, therefore, direct comparisons with 'per-instance' methods cannot be made. Some earlier hyper-heuristic methods do attempt a similar kind of generalisation (e.g. [10]), however as we have already shown in [13] that the original version of NELLI outperforms these methods we omit these comparisons.

In addition to comparing to NELLI [13], we compare our results to those obtained by a set of four well known heuristics from the literature (FFD, DJD, ADJD and DJT, see [16]) in which the best heuristic for each problem is selected using a greedy approach. Additionally, where appropriate we also compare to our own earlier work using an AIS model introduced in [17] and an island-model evolutionary algorithm described in [14].

Table 1. Default parameter settings for experiments

Parameter	Description	Value
n_p	number of problems added each iteration	30
n_h	number of heuristics added each iteration	1
c_{init}	initial concentration of heuristics/problems	200
Δ_c	variation in concentration based on stimulation	50
c_{max}	maximum concentration level	1000
p_m	Probability of mutation	0.75
l_{max}	maximum initial heuristic sequence length	10

5.1 Generalisation Capabilities

In order to test the generalisation capabilities of NELLI*, in the first instance the full set of 1370 instances was randomly split into two equally sized sets, labelled *train* and *test* — each set contained an equal distribution of examples from each of the 5 constituent problem sets. NELLI* was run for 200 iterations using the *train* set before being presented with the unseen problems in the *test* set. Performance was evaluated in terms of the number of problems solved optimally and in terms of the number of bins required over the known optimal solution and is shown in table 2 where all results are averaged over 20 runs.

NELLI* clearly generalises, finding the optimal solution to 82.6% of the unseen problems, and reducing the number of bins over optimal in comparison to the other algorithms. Comparing the results obtained using NELLI and NELLI* using a t-test proves the result is highly significant ($P < 0.0001$). Although not shown, NELLI* continues to learn and improve results (albeit at a slower rate) if executed over a much larger number of generations.

An additional experiment was performed in which every 200 iterations, a random set of n instances were selected from $\mathcal{U}$ to form the environment $\mathcal{E}$ and presented to the system. At each iteration, the performance of the system against $\mathcal{U}$ (the complete universe of 1370 problems) is measured. Recall that

Table 2. Results on the unseen 685 problems in the test set after 200 iterations training on the 685 problems in the training set

	Problems Solved				Extra Bins			
	min	max	mean	sd	min	max	mean	sd
Greedy Heuristic selection	548	548	548	0	188	188	188	0
AIS model [17]	554	559	556	1.4	159	165	162	1.4
Island Model [14]	552	559	557	1.4	159	164	162	1.4
NELLI [13]	559	559	559	0	159	159	159	0
NELLI*	549	**576**	**566**	5.8	**131**	164	**146**	8.2

at each iteration, the environment contains at most $n = 685$ problems, and only $n_p = 30$ of these are presented to the network at each instance, hence many of the instances in $\mathcal{U}$ have never been seen. Figure 3 plots performance over $\mathcal{U}$ over 1000 iterations (averaged over 20 runs) for $n \in 100, 200, 685$ and compares the results to the best result obtained by the old version of NELLI. A number of important points are apparent: NELLI* clearly outpeforms NELLI; it generalises over $\mathcal{U}$ — recall that in the early iterations most of the problems in $\mathcal{U}$ are unseen; it continues to learn over time; the ability to generalise is maximised by increasing the size of $\mathcal{E}$.

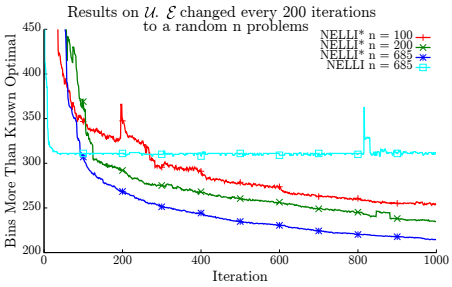


Fig. 3. The average number of bins greater than the optimal; the environment $\mathcal{E}$ is replaced with n randomly selected problem instances every 200 iterations

5.2 Memory and Learning

In order to investigate the memory capabilties of NELLI* we conduct an experiment in which the environment $\mathcal{E}$ is toggled between two different datasets every 200 iterations. The first dataset contains the 720 problems in $ds1$ and the second the 480 problem instances in $ds2$. As previously mentioned, these datasets have different characteristics such that heuristics that perform well on one dataset are not expected to perform well on the second. Typically, $ds1$ problems are also easier to solve. Each dataset is presented twice in the sequence $ds2, ds1, ds2, ds1$ following a 'start-up' epoch in which the system is initialised from scratch with $ds1$ and run for 200 iterations. For each dataset, we record the number of bins

more than optimal at the start of each epoch it is introduced (b_s), and at the end of each epoch (b_e). We formulate the following hypotheses:

- *Hypothesis 1* if the system has retained some memory of heuristics that previously solved these problems, we expect the value of b_e at epoch t to be similar to b_s at the next epoch the dataset is introduced (epoch $t+2$)
- *Hypothesis 2* If the system continues to learn over an epoch, there *should* be a significant difference between b_s and b_e over an epoch during which the dataset is present
- *Hypothesis 3* If the system continues to learn over its lifetime, there *should* be a significant difference between b_e at the first epoch the dataset appears, and b_e at the last epoch it occurs

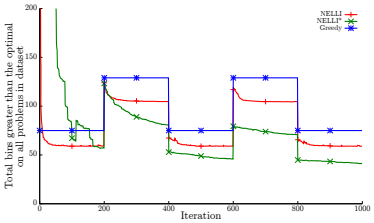
The results are shown in Table 3 and graphically in figure 4a, averaged over 20 runs in each case and compared to the results previously published in [17]. With respect to *hypothesis 1*, t-tests conducted on the values obtained at the end of epoch 1 and the start of epoch 3 for $ds2$, and epochs 2 and 4 for $ds1$ show no significant difference between results (P=0.581,0.581), thus we infer that the system has retained memory[1]. With respect to *hypothesis 2*, t-tests between the values of b_s and b_e at the two epochs where $ds2$ appears both given p-values < 0.0001, confirming that the observed difference is performance is significant. The same result is found for $ds1$ at epochs 2 and 4. Thus, we confirm that learning occurs over an epoch. Finally, we compare the value of b_e at epoch 1 with b_e at epoch 3 ($P < 0.0001$) and similarly with epochs 2 and 4 ($P = 0.0138$) proving the ability of NELLI* to learn over time.

Figure 3 confirms these trends by further analysing the final experiment described in section 5.1 in which $\mathcal{E}$ is changed to a randomly selected set of 685 instances from the 1370 problems in $\mathcal{U}$ every 200 instances. The same general trends are observed as in Figure 4a in that learning continues across the 1000 iterations. The difference between b_e and b_s is less defined at each epoch in this instance, as problems in $\mathcal{E}$ at epoch t are likely to overlap with those in $\mathcal{E}$ at epoch $t+1$. The magnitude of $|b_e - b_s|$ (where b_e is measured at the end of epoch t and b_s at the start of epoch $t+1$ decreases over time, as a direct result of the *memory* of the system.

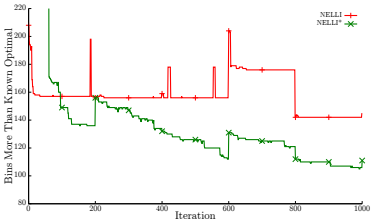
Table 3. Bins greater than the known optimal at epochs. Averaged over 20 runs

	Epoch 1 DS2		Epoch 2 DS1		Epoch 3 DS2		Epoch 4 DS1	
Set	Start	End	Start	End	Start	End	Start	End
NELLI* $\mathcal{E}$	123.28	80.61	52.94	45.78	79.39	70.72	44.83	41.28
NELLI $\mathcal{E}$	123.95	104.8	67.3	59	116.95	104.65	65.6	59
Greedy $\mathcal{E}$	129	129	75	75	129	129	75	75
NELLI* $\mathcal{U}$	312.89	259.17	257.67	247.22	246.89	233.94	234.44	229.17
NELLI $\mathcal{U}$	333.1	315.75	317.05	321.5	320.95	315.15	315.25	320.85
Greedy $\mathcal{U}$	364	364	364	364	364	364	364	364

[1] Strictly speaking, this only suggests that there is no evidence to suggest otherwise rather than providing proof.



(a) The average *total* number of bins greater than the optimal alternating every 200 iterations between ds1 (summed over 720 problems) and ds2 (summed over 480 problems).



(b) $\mathcal{E}$ is changed every 200 iterations to a random 685 problems taken from $\mathcal{U}$. The graph shows the average *total* number of bins greater than the optimal summed over the 685 problems

Fig. 4. Performance on alternating datasets, averaged over 20 runs

6 Conclusions and Future Work

In an extension to previous work, we have shown that the NELLI* system is capable of operating as a life-long learning (LML) system. As identified by [12] in their recent proposal, the system exhibits the three defining characteristics of an LML system: it incorporates a long-term memory; it selectively transfers prior knowledge when learning new tasks; it adopts a systems approach that ensures the effective and efficient interaction of the elements of the system. In comparison to previous hyper-heuristic approaches, it obtains better performance when evaluated according to two metrics, number of problems solved, and number of bins over optimal. Although any hyper-heuristic method that focuses on solving large sets of problems will inevitably trade some loss in performance against both generality and speed when compared to approaches that optimise solutions for each problem individually, we believe that in practice, such solutions are more than acceptable. In a recent article considering why evolutionary algorithms are not widely adopted in the real-world [8], the author notes that in industry, organisations do not have time to generate globally optimum solutions and therefore place higher importance on finding robust, quality solutions that can be generated quickly due to changes in the environment. NELLI directly addresses this issue, in providing cheap, high quality solutions in a system that does not need either tuning or modifying even as the environment it operates in changes.

References

1. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: A survey of the state of the art. J. Oper. Res. Soc. (July 2013)
2. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automating the packing heuristic design process with genetic programming. Evol. Comput. 20(1), 63–89 (2012)
3. Falkenauer, E.: A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics 2, 5–30 (1996)

4. Garrido, P., Riff, M.C.: Collaboration between hyperheuristics to solve strip-packing problems. In: Melin, P., Castillo, O., Aguilar, L.T., Kacprzyk, J., Pedrycz, W. (eds.) IFSA 2007. LNCS (LNAI), vol. 4529, pp. 698–707. Springer, Heidelberg (2007)
5. Gent, I.P.: Heuristic solution of open bin packing problems. Journal of Heuristics 3(4), 299–304 (1998)
6. Jackson, D.: Single node genetic programming on problems with side effects. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 327–336. Springer, Heidelberg (2012)
7. Maturana, J., Lardeux, F., Saubion, F.: Autonomous operator management for evolutionary algorithms. Journal of Heuristics 16, 881–909 (2010)
8. Michalewicz, Z.: Ubiquity symposium: Evolutionary computation and the processes of life: The emperor is naked: Evolutionary algorithms for real-world applications. Ubiquity 2012(November), 3:1–3:13 (2012)
9. Remde, S., Cowling, P., Dahal, K., Colledge, N., Selensky, E.: An empirical study of hyperheuristics for managing very large sets of low level heuristics. J. Oper. Res. Soc. 63(3), 392–405 (2012)
10. Ross, P., Schulenburg, S., Marin-Blazquez, J.G., Hart, E.: Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2002, pp. 942–948 (2002)
11. Scholl, A., Klein, R., Jürgens, C.: Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Comput. Oper. Res. 24(7), 627–645 (1997)
12. Silver, D., Yang, Q., Li, L.: Lifelong machine learning systems: Beyond learning algorithms. AAAI Spring Symposium Series (2013)
13. Sim, K., Hart, E., Paechter, B.: A lifelong learning hyper-heuristic method for bin packing. Evolutionary Computation Journal (in press, January 2014)
14. Sim, K., Hart, E.: Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In: Blum, C. (ed.) GECCO 2013: Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference, ACM, New York (2013)
15. Sim, K., Hart, E.: An improved immune inspired hyper-heuristic for combinatorial optimisation problems. In: GECCO 2014: Proceeding of the Sixteenth Annual Conference on Genetic and Evolutionary Computation Conference (in press, 2014)
16. Sim, K., Hart, E., Paechter, B.: A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part II. LNCS, vol. 7492, pp. 348–357. Springer, Heidelberg (2012)
17. Sim, K., Hart, E., Paechter, B.: Learning to solve bin packing problems with an immune inspired hyper-heuristic. In: Advances in Artificial Life, ECAL 2013: Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems, pp. 856–863. MIT Press (2013)
18. Thabtah, F., Cowling, P.: Mining the data from a hyperheuristic approach using associative classification. Expert Systems with Applications 34(2), 1093–1101 (2008)
19. Trojanowski, K., Wierzchon, S.T.: Immune-based algorithms for dynamic optimization. Information Sciences 179(10), 1495–1515 (2009)
20. Whitbrook, A.M., Aickelin, U., Garibaldi, J.M.: Two-timescale learning using idiotypic behaviour mediation for a navigating mobile robot. Appl. Soft Comput. 10(3), 876–887 (2010)