A Lifelong Learning Hyper-heuristic Method for Bin Packing

Kevin Sim Emma Hart Ben Paechter

Abstract

We describe a novel Hyper-heuristic system which continuously learns over time to solve a combinatorial optimisation problem. The system continuously generates new heuristics and samples problems from its environment; representative problems and heuristics are incorporated into a self-sustaining network of interacting entities inspired by methods in Artificial Immune Systems.The network is plastic in both its structure and content leading to the following properties: it exploits existing knowledge captured in the network to rapidly produce solutions; it can adapt to new problems with widely differing characteristics; it is capable of generalising over the problem space. The system is tested on a large corpus of 3968 new instances of 1D-bin packing problems as well as on 1370 existing problems from the literature; it shows excellent performance in terms of the quality of solutions obtained across the datasets and in adapting to dynamically changing sets of problem instances compared to previous approaches. As the network self-adapts to sustain a minimal repertoire of both problems and heuristics that form a representative map of the problem space, the system is further shown to be computationally efficient and therefore scalable.

1 Introduction

The previous two decades have seen significant advances in meta-heuristic optimisation techniques that are able to quickly find optimal or near-optimal solutions to problem instances in many combinatorial optimisation domains. Techniques employed vary widely: typical meta-heuristic algorithms (e.g. evolutionary algorithms, particle swarm optimisation) operate by searching a space of potential problem solutions. Hyper-heuristic algorithms on the other hand operate by searching a space of heuristics which are used to either perturb existing solutions or construct completely new solutions. Despite many successful applications of both these approaches, they typically operate in the same manner: an algorithm is tuned to work well on a (possibly large) set of representative problems and each time a new problem instance needs to be solved, the algorithm conducts a search of either the solution space or the heuristic space to locate good solutions. Whilst this often leads to acceptable solutions, such approaches have a number of weaknesses in that if the nature of the problems to be solved changes over time, then the algorithm needs to be periodically re-tuned. Furthermore, such approaches are likely to be inefficient, failing to exploit previously learned knowledge in the search for a solution.

In contrast, in the field of machine-learning, several contemporary learning systems employ methods that use prior knowledge when learning behaviours in new, but similar tasks, leading to a recent proposal from Silver et al. (2013) that *it is now appropriate for the AI community to move beyond learning algorithms to more seriously consider the nature of systems that are capable of learning over a life- time*. They suggest that algorithms should be capable of learning a variety of tasks over an extended period of time such that the knowledge of the tasks is retained and can be used to improve learning in the future. They name such systems *lifelong machine learning*, or *LML* systems, in accord with earlier proposals by Thrun and Pratt (1997). Silver et al. (2013) identify three essential components of an LML system: it should be able to retain and/or consolidate knowledge, i.e. incorporate a long-term memory; it should selectively transfer prior knowledge when learning new tasks; it should adopt a systems approach that ensures the effective and efficient interaction of the elements of the system. In terms of the memory of the LML, they further specify that the system should be computationally efficient when storing learned knowledge in long-term memory and that ideally, retention should occur online.

Silver et al. (2013) propose a framework for a generic LML that encompasses supervised, unsupervised, and reinforcement learning techniques, with a view to developing test applications in the robotics and agents domains. In contrast, we turn to biology for inspiration in building an LML for optimisation purposes, and in particular to the natural immune system, noting that it has properties that fulfil the three requirements for an LML system listed above. It exhibits memory that enables it to respond rapidly when faced with pathogens it has previously been exposed to; it can selectively adapt prior knowledge via clonal selection mechanisms that can rapidly adapt existing antibodies to new variants of previous pathogens and finally, it embodies a systemic approach by maintaining a repertoire of antibodies that collectively cover the space of potential pathogens.

Using this analogy, we describe an LML system for optimisation that combines inspiration from immunology with a Hyper-heuristic approach to optimisation, using 1D bin-packing as an example domain. The system continuously generates new knowledge in the form of novel deterministic heuristics that produce solutions to problems; these are integrated into a network of interacting heuristics and problems: the problems incorporated in the network provide a minimal representative map of the problem space; the heuristics generalise over the problem space, each occupying its own niche. Memory is encapsulated in the network and is exploited to rapidly find solutions to new problems. The network is plastic both in its contents and its topology, enabling it to continuously adapt as the problems in its environment change. We show that system not only produces effective solutions, but also responds efficiently to changing problems in terms of the response time required to obtain an effective solution.

2 Previous Related Work

We describe an *LML* for bin-packing which combines inspiration from both Hyper-heuristics and Artificial Immune Systems with the concept of LML set out by Silver et al. (2013). We briefly review relevant previous work in each of these domains that informs our approach, focusing on the aspects of each field that particularly relate to learning.

2.1 LML systems

Systems that learn over extended periods of time are common in the machine learning literature. Silver et al. (2013) describe several examples that cover supervised, unsupervised and reinforcement learning methods. Ruvolo and Eaton (2013) propose an Efficient Lifelong Learning Algorithm dubbed ELLA that focuses on multi-task learning in machine-learning applications relating to prediction and recogniton tasks; ELLA is able to transfer previously learned knowledge to a new task and integrate new knowledge through the use of shared basis vectors in a manner that is proved to be computationally efficient. Other examples are found in the multi-agent system (Kira and Schultz, 2006) and robotics literature (Carlson et al., 2010). The potential benefits to be gleaned by incorporating concepts taken from the meta-learning community into Hyper-heuristic approaches are explored in Pappa et al. (2013). Despite this, there appears to be little literature in the optimisation field. An exception is work originated by Louis and McDonnell (2004) on Case-Injected Genetic Algorithms — CIGAR algorithms. Conjecturing that a system that combined a robust search algorithm with an associative memory could learn by experience to solve problems, they combine a genetic algorithm with a case-based memory of past problem solutions. Solutions in the memory are used to seed the population of the genetic algorithm; cases are chosen on the basis that similar problems will have similar solutions, and therefore the the memory will contain building blocks that will speed up the evolution of good solutions. Using a set of design and optimisation problems, Louis and McDonnell (2004) demonstrate that their system learns to take less time to provide quality solutions to a new problem as it gains experience from solving other similar problems. The system differs from other case base reasoning systems in that it does not use prior knowledge based on a comparison to previously encountered problems but simply injects solutions that are deemed similar to those already in the GA population.

The CIGAR methodology can be considered as an LML technique as it builds up a case-history over time. It differs from the system proposed in this article in its use of genetic algorithms as the optimisation technique rather than a Hyper-heuristic approach, and in its assumption that similar problems will have similar solutions; this limits the generality of the system to those where the similarity metric between solutions can be easily defined. In addition, the memory in a CIGAR algorithm does not adapt over time, for example enabling 'forgetting', but simply increases in size as time passes.

2.2 Hyper-heuristics

A Hyper-heuristic is an automated method for selecting or generating heuristics to solve hard computational search problems, motivated by a desire to raise the level of generality at which search methodologies can operate (Burke et al., 2003). Rather than typical meta-heuristic methodologies which tend to be customised to a particular problem or a narrow class of problems, the goal of Hyper-heuristics is to develop algorithms that work well across many instances from a problem class and do not require parameters to be tuned before use. Although this may lead to some trade-off in quality compared to specifically designed meta-heuristic approaches, Hyper-heuristics should be fast and exhibit good performance across a wide range of problems (Ross, 2005).

A recent categorisation of Hyper-heuristics by Burke et al. (2010a) separated Hyper-heuristics into two categories — heuristic *selection*, i.e. choosing between existing heuristics, and heuristic *generation*, i.e. creating new heuristics from components of existing ones. The same authors further classify Hyper-heuristics according to type of learning used to inform the search process. Thus, according to their definition, in an *online* Hyper-heuristic, learning takes place while the algorithm is solving an instance of a problem. In contrast, *offline* Hyper-heuristics learn from a representative set of training instances resulting in a model that generalises to unseen problem instances. However both categorisation suffer from the same weakness; learning ends at some point. In online systems, every time a new instance is solved, the learning process starts from scratch. Although off-line systems do generalise from a set of problems, they need to be periodically re-trained if the nature of the problems changes.

With respect to heuristic generation, genetic programming has commonly been used in an offline learning procedure to generate a set of novel heuristics that work well on a training set and are expected to generalise to unseen instances, e.g. (Burke et al., 2010b, 2009, 2012). However, research on *disposable* heuristics has also been conducted (Bader-El-Den and Poli, 2008) that evolves novel heuristics for solving a single instance of a problem; this type of approach is in direct conflict to the appeal from Silver et al. (2013) to develop LML systems. Previous studies in the domain of bin packing (Burke et al., 2007b; Terashima-Marín et al., 2010) have strengthened the notion that whilst it is possible to create heuristics that generalise well across a range of problem instances there is a trade off between the generality of a heuristic and the solution quality it provides. The approach taken here is to generate sets of complementary heuristics that can be seen to generalise well across the complete set of problem instances whilst being individually tailored to niche areas of the problem space.

Finally, it is worth mentioning an approach from Sim and Hart (2013) in which heuristic generation is combined with a greedy selection algorithm in a learning system inspired by island models in evolution, to learn a set of heuristics that collaborate to solve representative sets of problem instances. This work differs from much previous Hyper-heuristic work in trying to learn an appropriate set of heuristics rather than learn to select from a large but fixed set of heuristics, but suffers from the same criticisms that have been levelled at other offline Hyper-heuristic methods in that the learning phase must be repeated if the nature of the problem instances changes.

2.3 Immune Networks

As described in Section 1, the immune system exhibits many of the properties and functions desired in LML systems. Many mechanisms have been proposed as to how such functions are achieved, with no clear consensus emerging amongst the immunological community. Of most relevance to this work is the idiotypic network model proposed by Jerne (1974) as a possible mechanism for explaining long-term memory. Challenging existing thinking at the time, Jerne proposed that antibodies produced by the immune system interact with each other, even in the absence of pathogens, to form a self-sustaining network of collaborating cells, that collectively embodies a memory of previous responses. Bersini (1999) proposed that the engineering community might benefit from developing algorithms inspired by double plasticity of the network view of the immune system: *parametric* plasticity provides an adaptive mechanism that adjusts parameters while executing a task to improve performance while *structural* plasticity enables new elements to be incorporated into network and elements to be removed, thereby enabling the network to adapt to a time-varying environment — properties which result in an LML system.

Idiotypic network theory has been translated into a number of computational algorithms in machine-learning, optimisation and engineering domains (see (Timmis, 2007) for an overview). However, very few of these applications really address problems in the kind of complex, dynamic environments envisaged by Bersini (1999). Nasraoui et al. (2003) describe a system based on idiotypic network for tracking evolving clusters in noisy data-streams, finding it to be both capable of learning and scalable. However, the majority of work in relevant dynamic environments lies in the robotics domain.

Idiotypic networks were first used in mobile robotics in (Watanabe et al., 1998), where parametric plasticity was exploited to enable a robot to adapt its behaviours in order to fulfil a task, depending on environmental conditions. An antibody in the network described a tuple $< condition, action >$: conditions match environmental data, and the action specifies an atomic behaviour of the robot that should be executed if the antibody has high enough concentration. Connections between antibodies either further stimulate or suppress antibodies, altering their concentration and therefore their probability of selection. Early work relied on hand-coded antibodies within the network, and focused on learning connections. This was significantly improved by Whitbrook et al. (2007, 2008, 2010) who used an evolutionary algorithm in separate learning phase to produce antibodies which are used to seed the network. Despite the this, the network does not have structural plasticity — the initial learning phase happens only once and hence the networks cannot adapt to significant environmental changes.

Although immune-inspired algorithms are common in the field of optimisation, the majority are applied to static optimisation problems, e.g. (Kromer et al., 2012). However, some research exists relating to dynamic optimisation in which the fitness function of a problem changes over time. Nanas and de Roeck (2007) give a comprehensive overview of immune-inspired research in this area, with Trojanowski and Wierzchon (2009) providing an detailed analytical comparison on a series of benchmarks. For example, in (Gaspar and Collard, 2000; F.O. de Frana, 2005), idiotypic network inspiration results in a dynamic allocation of population size; these algorithms can thus be said to exhibit structural plasticity, in that the network itself selects which recruited nodes will survive in the population, essentially removing similar nodes to preserve diversity. Idiotypic effects also implicitly provide memory in these algorithms, by sustaining nodes within the network.

2.4 Immune Networks and Hyper-heuristics

An initial attempt at combining Hyper-heuristics with immune-inspiration from immune network theory to evolve a collaborative set of heuristcs for solving optimisation problems was described by Sim et al. (2013). To the best of our knowledge, this was the first example of an immune-inspired Hyper-heuristic optimisation system and was tested using 1370 1D bin-packing problems. This system used methods from single-node genetic programming (SNGP) (Jackson, 2012a,b) to generate a stream of novel heuristics. If one heuristic H_a could solve a problem from a set of problems of interest using fewer bins than another heuristic H_b then H_a received a positive stimulation, proportional to the difference in bins used. Heuristics with zero stimulation were removed from the system, resulting in a network of heuristics in which a direct measure of interaction could be calculated between every pair of heuristics. Although the system provided promising results, it had a number of drawbacks. Firstly, the algorithm required each heuristic in the system to be evaluated against all of the problems in the set of interest, which can be computationally costly for large problem sets. Secondly, the algorithm required the use of a greedy procedure to remove heuristics that

Table 1: Benchmark bin-packing problems

Data Set	capacity (c)	n	ω	#Problems
ds_1	100,120,150	50,100,200,500	[1,100],[20,100],[30,100]	$36 \times 20 = 720$
ds_3	100000	200	[20000,30000]	10
$FalU$	150	120,250,500,1000	[20,100]	$4 \times 20 = 80$
$FalT$	1	60,120,249,501	[0.25,0.5]	$4 \times 20 = 80$

Data Set	c	n	ϖ (avg weight)	$\delta(\%)$	# Problems
ds_2	1000	50,100,200,500	$\frac{c}{3}, \frac{c}{5}, \frac{c}{7}, \frac{c}{9}$	20,50,90	$48 \times 10 = 480$

were subsumed by other heuristics in the system in order to limit the network size. However, this system inspired the work presented in this article, which both addresses limitations of previous work and extends it to create a continuous-learning optimisation system that is capable of learning over long periods of time in dynamic environments. Note that the term dynamic does not refer here to changing problem instances but to dynamically changing environments that are composed of varying sets of problem instances.

3 Problem Definition: 1D bin packing - benchmarks and heuristics

The objective of the one dimensional bin packing problem (BPP) is to find a packing which minimises the number of containers, b, of fixed capacity c required to accommodate a set of n items with weights $\omega_j : j \in \{1 \dots n\}$ falling in the range $1 \leq \omega_j \leq c, \omega_j \in \mathbb{Z}$ whilst enforcing the constraint that the sum of weights in any bin does not exceed the bin capacity c. The lower and upper bounds on b, (b_l and b_u) respectively, are given by Equation 1. Any heuristic that does not return empty bins will produce, for a given problem instance, p, a solution using b_p bins where $b_l \leq b_p \leq b_u$.

$$b_l = \left\lceil \sum_{j=1}^{n} \omega_j \div c \right\rceil , \; b_u = n \tag{1}$$

Studies relating to bin-packing generally use instances from one or all of the data-sets given in Table 1. In each of these sets, a number of problem instances are generated from a set of parameters, also shown in the table.

Data sets $ds1, ds2$ & $ds3$ were introduced by Scholl et al. (1997). All have optimal solutions that differ from the lower bound given by Equation 1. However all are known and have been solved since their introduction (Schwerin and Wäscher, 1997). $ds1$, $ds3$ were created by generating n items with weights randomly sampled from a uniform distribution between the bounds given by ω. $ds2$ was created by randomly generating weights from a uniform distribution in the range given by $\varpi \pm \delta$.

All of the instances from $FalU$ and $FalT$, introduced by Falkenauer (1996), have optimal solutions at the lower bound except for one for which it has been proven not to exist (Gent, 1998). $FalU$ was created by generating n items with weights randomly sampled from a uniform distribution between the bounds given by ω. The instances in $FalT$ were generated such that the optimal solution has exactly 3 items in each bin with no free space.

In the remainder of the article, the complete set of problems described in this section obtained from the benchmark literature is referred to as problem set A. The suffix A_{ds_1} for example denotes problems from ds_1 (Table 1) in the benchmark set A. A number of deterministic heuristics are commonly used to solve these problems, and are particularly prevalent in the Hyper-heuristic literature. The heuristics are described in Table 2. Note that none of the heuristics listed in the table were able to find optimal solutions to any problems from the datasets $ds3$ or $FalT$. A number of other heuristics were examined including Best Fit Garey and Johnson (1979) and the Sum of Squares Csirik et al. (1999) algorithm but were excluded from this study as they provided no further improvement when evaluated on the problem sets used.

Table 2: Common deterministic heuristics from the literature for solving 1D bin-packing problems

Heuristic	Acronym	Summary
First Fit Descending	FFD	Packs each item into the first bin that will accommodate it. If no bin is available a new bin is opened.
Djang and Finch Djang and Finch (1998)	DJD	Packs items into a bin until it is at least one third full. The set of up to three items which best fills the remaining space is then found with preference given to sets with the lowest cardinality. The bin is then closed and the procedure repeats using a new bin.
DJD more Tuples Ross et al. (2002)	DJT	works as for DJD but considers sets of up to five items after the bin is filled more than one third full.
Adaptive DJD Sim et al. (2012)	ADJD	packs items into a bin until the *free space* is less than or equal to three times the average size of the remaining items. It then operates as for DJD.

3.1 Additional Novel Problems

In order to evaluate the system on a larger set of problems a generator was implemented that facilitates generation of problem instance with similar characteristics to those described previously. The generator (Sim, 2013) attempts to generate 3 new problem instances from parameters obtained from each of the problems in set A but tries to enforce an additional constraint that the total free space summed across all bins is zero, thus increasing the complexity of the problem[1]. $(1370 \times 3 = 4110)$ new problems were initially generated, however, not of all the generated instances respected the free space constraint. 142 instances in which the total free space was greater than 1 bin were removed. Of the remainining 3968 new instances, 3178 had optimal solutions where all bins were filled to capacity. The remaining 790 had optimal solutions at the lower bound given by Equation 1 where the free space summed across all bins was less than the capacity of one bin. The problem instances can be downloaded from the internet (Sim, 2013) along with a known optimal solution for each.

In the remainder of the article, this new problem set is denoted *Problem Set B*. A problem described as being from B_{ds_1} denotes a novel instance generated from parameters derived by sampling the corresponding problem instances from Problem Set A_{ds1} Scholl et al. (1997).

4 An LML Hyper-heuristic

The LML Hyper-heuristic system proposed comprises of three main parts: a stream of problem instances, a heuristic generator and an AIS, as illustrated by Figure 1. The system is dubbed $NELLI$ - **NE**twork for **L**ife **L**ong learn**I**ng.

NELLI is designed to run continuously; problem instances can be added or removed from the system at any point. A heuristic generator akin to gene libraries in the natural immune system provides a continual source of potential heuristics. The AIS itself consists of a network of interacting problems and heuristics (akin to immune cells in the natural immune system) that interact with each other based on an affinity metric.

The immune network sustains a minimal repertoire of heuristics *and* a minimal repertoire of problems that provide a representative map of the problem space to which the system has been exposed over its lifetime. From a problem perspective, the network does not contain representatives of *all* problems from the problem stream shown in Figure 1, but a representative set that is sufficient to map the problem space. From a heuristic perspective, only heuristics that provide a unique contribution in that they produce a better result on at least one problem than any other heuristic are retained.

This is represented conceptually in Figure 2. In this diagram, the Figure (a) shows a set of problems $\mathcal{E}$ that the system is currently exposed to. Figure (b) shows a set of heuristics $\mathcal{H}$ that collectively cover the

[1]The problems instances generated prove harder for the benchmark deterministic heuristics to solve optimally as can be seen by comparing Tables 7 and 9

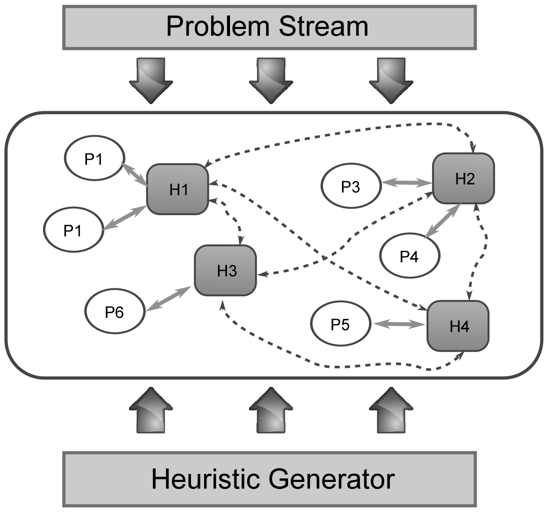


Figure 1: A conceptual view of the system: problems are continuously added/removed from the system. The generator continuously injects new heuristics. The dynamics and meta-dynamics of the system result in a self-sustaining network of heuristics and problems. Solid lines show *direct* interactions, dashed lines represent *indirect* interactions (see Section 4.2)

problems in $\mathcal{E}$. The problems $P1$ and $P2$ are solved equally by two or more heuristics. H2 is subsumed in that it cannot solve *any* problem better than another heuristic. In Figure (c), H2 is removed as it does not have a niche in solving problems; problems P1 and P2 are removed as they do not have a niche in describing the problem space[2]. A competitive exclusion effect is observed between heuristics (and also between problems) that results in efficient coverage of the problem space. A key aspect of the compression is that it significantly decreases the computation time of the method (discussed in more detail in Section 5.2.2). The mechanism by which this is achieved is described later in Section 4.1. Finally, meta-dynamic processes continuously generate novel heuristics and adapt the network structure. Thus, the system has the following features:

- It rapidly produces solutions to new problems instances that are similar in structure to previous problem instances that the system has been exposed to
- It responds by generating new heuristics to provide solutions to new problems that differ from those previously seen

In the next sections, we describe the key components of the LML system.

4.1 The Artificial Immune System

The AIS component is responsible for constructing a network of interacting heuristics and problems, and for governing the dynamic processes that enable heuristics to be incorporated or rejected from the current network. Pseudo-code describing the network dynamics is given in Algorithm 1. The following variables are used in the algorithm definition and in the remainder of the article.

- $\mathcal{U}$ - the set of all possible problems from the class of 1D-BPP of interest.
- $\mathcal{U}'$ - a subset of $\mathcal{U}$ that contains a specific set of problems, e.g problem set A or B

[2]although these problems have been removed from the network, they can still be solved by the system as heuristics H1 and H3 remain in the network

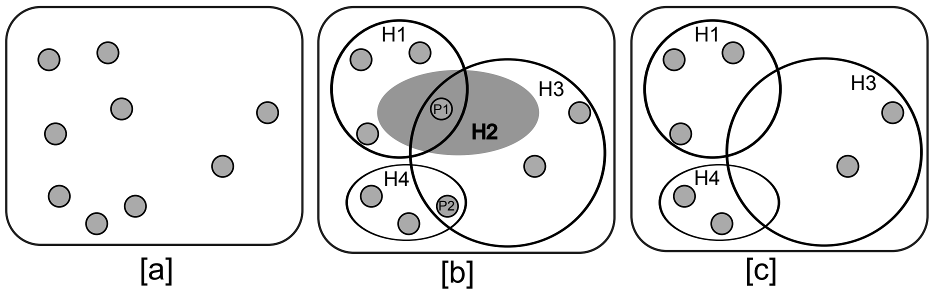


Figure 2: Diagram [a] shows the problems that the system is currently exposed to $\mathcal{E}$. The middle diagram [b] shows a set of generated heuristics that cover the problems in $\mathcal{E}$. The problems $P1$ and $P2$ shown are equally solved by one or more heuristics and therefore not required to map the problem space. The shaded heuristic is redundant as it does not have a niche. The right-hand diagram [c] shows the resulting *network* $\mathcal{N}$ that sustains the minimal set of problems and heuristics required to describe the space.

- $\mathcal{E}$ - the current environment, i.e. the set of problems we are currently interested in solving, i.e. $\mathcal{E} \subset \mathcal{U}' \subset \mathcal{U}$
- $\mathcal{E}^*$ - the set of *all* problems to which the system has been exposed during its lifetime
- $\mathcal{N}$ - the immune network, comprised of a set of problems and a set of heuristics
- $\mathcal{P}$ - the set of *problems* currently sustained in the immune network $\mathcal{N}$, i.e. $\mathcal{P} \subset \mathcal{E}^*$
- $\mathcal{H}$ - the set of *heuristics* currently sustained in the immune network $\mathcal{N}$

The algorithm captures the three essential concepts of an immune network as proposed by Varela et al. (1988) — structure, dynamics, and meta-dynamics. The term *structure* refers to the interactions between components of the network, in this case, problems and heuristics, and is described in Section 4.2 and by steps 4 and 5 of algorithm 1. *Dynamics* refers to the variations in time of the concentration and affinities between components of the network, and crucially, describes how the network adapts to itself and the environment (steps 6-8 in algorithm 1). Finally, the network *metadynamics* refers to a unique property of the immune system, that is the ability to continuously produce and recruit novel components, in this case, heuristics and problems, as described in Section 4.4 and step 8. These elements are discussed in detail in the next sections.

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 $t$
 1: repeat
 2:     optionally replace $\mathcal{E}$ : $\mathcal{E}^* \leftarrow \mathcal{E}^* \cup \mathcal{E}$
 3:     Add $n_h$ randomly generated heuristics to $\mathcal{H}$ with concentration $c_{init}$
 4:     Add $n_p$ randomly selected problem instances from $\mathcal{E}$ to $\mathcal{P}$ with concentration $c_{init}$
 5:     calculate $h_{stim} \forall h \in \mathcal{H}$ using Equation 2
 6:     calculate $p_{stim} \forall p \in \mathcal{P}$ using Equation 3
 7:     increment all concentrations (both $\mathcal{H}$ and $\mathcal{P}$) that have concentration $< c_{max}$ and stimulation $> 0$ by $\Delta_c$
 8:     decrement all concentrations (both $\mathcal{H}$ and $\mathcal{P}$) with stimulation $\leq 0$ by $\Delta_c$
 9:     Remove heuristics and problems with $concentration \leq 0$
10: until $stopping\ criteria\ met$
```

4.2 Network Structure

The network $\mathcal{N}$ sustains a set of interacting heuristics and problems. Problems are *directly* stimulated by heuristics, and vice versa. Heuristics are *indirectly* stimulated by other heuristics.

A heuristic h can be stimulated by one or more problems. 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 expressed mathematically by Equation 2, in which $\mathcal{H}'$ is the set of heuristics currently in the system, excluding the heuristic h currently under consideration, i.e. $\mathcal{H}' = \mathcal{H} - h$.

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

Note that heuristics are *directly* stimulated by problems. A heuristic only survives if it is able to solve at least one problem better than any other heuristic in the system. This provides competition between heuristics which forces a heuristic to find an individual niche to ensure survival. Thus, although no quantitative value is calculated for heuristic$\leftrightarrow$heuristic interactions, we consider this an *indirect* interaction arising from the method of calculating the problem$\leftrightarrow$heuristic interactions.

As the affinity between a problem and a heuristic is symmetrical, then the stimulation of a problem is simply the affinity between the problem and the heuristic that best solves it. A problem for which the best solution is provided by more than one heuristic receives zero stimulation. Thus, unlike heuristics, a problem can only be stimulated by one heuristic. This is expressed mathematically in Equation 3. Note that in the sum expressed in this equation, at most one term will be non-zero.

$$p_{stim} = \sum_{h \in \mathcal{H}} \delta_{bins} \begin{cases} \delta_{bins} = \min(bins_{\mathcal{H}'_p})) - bins_{h_p} & : \text{ if } \min(bins_{\mathcal{H}'_p}) - bins_{h_p} > 0 \\ \delta_{bins} = 0 & : \text{ otherwise} \end{cases} \tag{3}$$

4.3 Network Dynamics

Each iteration, the environment $\mathcal{E}$ optionally changes (step 2). This can range from adding new problem instances to the current environment to completely replacing the current environment with a new set of problem instances. In step 3, one new heuristic is generated and is made available to the network. The affinity metric described encourages *diversity* in the network, in sustaining heuristics that cover different parts of the problem space. In a practical application however, it is reasonable to assume that in addition to maintaining diversity, important goals of the system should be to find (1) the set of heuristics that most efficiently cover the problem space and (2) the set that collectively minimise the total number of bins used to solve all problems the network is exposed to[3]. While the latter is addressed by sustaining any heuristic with non-zero affinity, the former goal requires some attention.

Previous AIS models relating to idiotypic networks generally make use of an equation first defined by Farmer et al. (1986) to govern the dynamics of addition and removal of nodes from a network. In machine-learning applications such as data-clustering this was quickly found to lead to population explosion (Timmis et al., 2000), later addressed by using resource limiting mechanisms (Timmis and Neal, 2001). In previous robotic applications, the situation is avoided completely by using a network of fixed size and focusing only on evolving connections. In more theoretical models (Hart, 2006) the criteria are not relevant, as the goal is simply to show that a network can be sustained. In this heuristic case, simply sustaining all

[3]in fact, exactly the same goals were identified in generic form by Bersini (1999)

heuristics that contribute to covering the heuristic space is likely to lead to population explosion in the same manner observed in data-mining applications, as no pressure exists on the system to encourage efficiency.

4.4 Network Meta-Dynamics

The proposed LML system requires a continuous source of novel heuristics to be generated. Burke et al. (2009) describe the use of Genetic Programming to generate new heuristics within a Hyper-heuristic framework; this has been applied specifically to bin-packing (Burke et al., 2006, 2007a). Although achieving some success, the approaches suffer from the usual afflictions of GP, in that efforts must be made to control unnecessary bloat. Sim and Hart (2013) proposed the use of Single Node Genetic Programming (SNGP) (Jackson, 2012a,b) as an alternative method of generating new bin-packing heuristics.

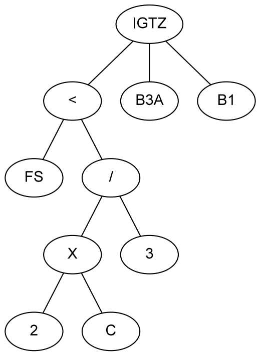
SNGP differs from the conventional GP model introduced by Koza (1992) in a number of key respects:

- Each individual node may be the starting point for evaluation, not only the top most node.
- Nodes may have any number of parent nodes (including none and duplicates) allowing for network structures other than trees to be formed.
- No crossover is used, only mutation which is employed as a hill climber with the mutation undone if no improvement is achieved.

The key benefit of this method is that the heuristics are of fixed maximum size (in terms of the number of nodes). Sim and Hart (2013) showed that SNGP could successfully evolve new bin-packing heuristics that outperformed existing ones from the literature. In this study, we only use the initialisation procedure of the SNGP method to produce new heuristics, and *do not* apply evolutionary operators to improve the generated heuristics. The justification for this is as follows: the role of the heuristic generator is to provide a continuous source of novel material for potential integration into the network of heuristics. The network dynamics will eradicate poor heuristics, and furthermore, given the relatively small number of terminal and function nodes outlined in Table 3, heuristics of reasonable quality are likely to be generated at random. Finally, it is more efficient to improve heuristics via an evolutionary operator only once they become established in the network, thereby proving their potential.

Figure 3 shows an example of a hand crafted heuristic represented in the SNGP format[4] — this is in fact the deterministic heuristic DJD. A complete automatically initialised SNGP structure is depicted in Figure 4. A fixed set of terminal and function nodes are available to the generator and are defined in Table 3 which combines nodes according to the process outlined in Algorithm 2. The nodes selected for use in this study were derived by examining the heuristics outlined in Table 2. The simplest of these heuristics, FFD, packs each item in turn, taken in descending order of item size, into the first bin with free space that will accommodate it. FFD can be represented by a single node B1. The other heuristics used for comparison can all be represented as tree structures similar to that depicted for DJD in Figure 3. Further justification for the choice of nodes and details of SNGP can be found in (Sim and Hart, 2013).

[4]In this case this is also a standard GP tree



The tree structure is repeatedly evaluated from the top IGTZ node until it fails to pack any more items into the current bin at which time a new bin is opened.

The left conditional child branch checks to see if the free space is less than 2 times the bin capacity divided by 3.

If this is true the second branch is evaluated and packs the best 3 items into the current bin.

If false the third branch is evaluated and the B1 node is executed and packs the single best item into the bin.

Figure 3: DJD Heuristic Expressed as a Tree

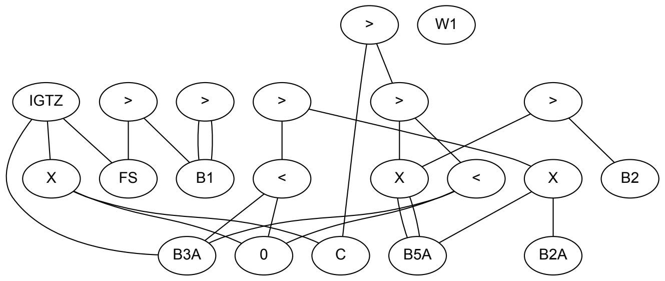


Figure 4: A randomly initialised SNGP structure

Table 3: Nodes Used

Function Nodes

/	Protected divide returns -1 if denominator is 0 otherwise the result of dividing the first operand by the second
$>$	Returns 1 if the first operand is greater than the second or -1 otherwise
IGTZ	Evaluates the first operand. If it evaluates as greater than zero the result of evaluating the second operand is returned otherwise the result of evaluating the third operand is returned
$<$	Returns 1 if the first operand is less than the second or -1 otherwise
X	Returns the product of two operands

Terminal Nodes

B1	Packs the single largest item into the current bin returning 1 if successful or -1 otherwise
B2	Packs the largest combination of exactly 2 items into the current bin returning 1 if successful or -1 otherwise
B2A	Packs the largest combination of up to 2 items into the current bin giving preference to sets of lower cardinality. Returns 1 if successful or -1 otherwise
B3A	As for B2A but considers sets of up to 3 items
B5A	As for B2A but considers sets of up to 5 items
C	Returns the bin capacity
FS	Returns the free space in the current bin
INT	returns a random integer value $\in (-1, ..., +5)$
W1	Packs the smallest item into the current bin returning 1 if successful else -1

Algorithm 2 Heuristic Generation

```
1: Each of the terminal nodes $T \in \{t_1, \dots t_r\}$ are added exactly once. The terminal nodes are given an integer identification number ranging from $1 \dots r$.
2: A number, $n$, of function nodes are selected at random from the set of all function nodes $F \in \{f_1, \dots, f_s\}$ and given an identification number ranging from $r + 1, \dots$ to $r + n$. This allows for the possibility of duplicate function nodes within the population or for SNGP structures with function nodes omitted.
3: The function nodes have all their child nodes assigned at random from nodes with a lower id thus preventing any infinite looping.
4: A single node is chosen at random to be the root node.
```

4.5 Comparison to previous work

A brief comparison of NELLI to the system described in (Sim et al., 2013) was conducted to highlight specific differences and improvements.

The two algorithms both utilise SNGP to provide a stream of novel heuristics as input to the system. However, this is the only similarity. The main difference between the two algorithms lies in the structure and composition of the self-sustaining network. In the previous work, the network consisted only of interacting heuristics, in which a direct measure of affinity a was calculated between each pair of heuristics. This measure was asymmetric, i.e. $a_{ab} \neq a_{ba}$. In contrast, the network sustained by NELLI consists of interacting problems *and* heuristics. An explicit measure of interaction is calculated between problems and heuristics which is symmetric. However, heuristics only interact indirectly through an implicit effect that excludes heuristics that do not occupy a specific niche within the problem space. The method by which the concentration of both heuristics and problems is calculated also directly results in unecessary heuristics and problems being removed, minimising the size of the network and removing the need for the greedy

Parameter	Description	Value
n_p	number of problems added each iteration from $\mathcal{E}$	30
n_h	number of new heuristics added each iteration	1
c_{init}	initial concentration of added heuristics/problems	200
Δ_c	variation in concentration based on stimulation level	50
c_{max}	maximum concentration level	1000

Table 4: Default parameter settings for experiments

calculation performed in (Sim et al., 2013) that was required to remove redundant heuristics.

As a result of these improvements, NELLI brings significant advantages. In addition to maintaining a set of heuristics that collaborate to cover the problem space, it also maintains a mininal set of problems $\mathcal{P}$ that are representative of the problem space $\mathcal{E}$, thereby providing a "map" of the space. The minimal network sustained brings considerable efficiencies in computational cost; the heuristics in the NELLI network only need to be evaluated against the minimal set of problems $\mathcal{P}$ sustained rather than the complete set of problems of interest $\mathcal{E}$ — in (Sim et al., 2013), heuristics needed to be evaluated against everything in $\mathcal{E}$ at each iteration. This results in a system that is both efficient and scalable.

5 Experiments and Results

Experiments were conducted to test the following features of the system.

- The utility of the Hyper-heuristic system compared to single deterministic heuristics, similar Hyper-heuristic approaches that use collectives of heuristics, and the best known solutions for each of the problems.
- The elasticity and responsiveness of the network in terms of its ability to quickly adapt when presented with new unseen problem instances.
- The ability to continually learn whilst retaining memory of previously encountered problem instances.
- The efficiency and scalability of the system in maintaining knowledge using a minimal repertoire of network components.

Experiments were conducted using the model described by Algorithm 1 using data drawn from the two sets of data described in Section 3, problem sets A and B. Unless specifically stated the default parameters used for all experiments were as shown in Table 4. These parameters were set following an initial period of empirical investigation.

5.1 Utility of system in comparison to previous approaches

Before analysing the behaviour of NELLI as an LML system, the system is benchmarked on static problem sets to obtain an indication of the quality of results it provides. Comparisons to the benchmark human designed deterministic heuristics are provided. In terms of comparison to other Hyper-heuristic appropaches, we provide comparisons to the precursor of NELLI described in (Sim et al., 2013) and also to another system described in (Sim and Hart, 2013) in which an island-model of cooperative co-evolution was used to find a collaborative set of heuristics. As no other authors have used the same extensive set of problems as in this paper direct comparisons to other Hyper-heuristic approaches from the literature are difficult. The most comprehensive study available is carried out by Ross et al. (2003) who evaluate their Hyper-heuristic on a subset of 890 problem instances consisting of all of the problems from $ds1$, $ds3$, $FalU$ and $FalT$[5]. This study

[5]The authors do not include the 480 problem instances from $ds2$ which prove hard for the variations of DJD used

Table 5: Results obtained on a static dataset of 685 problems taken from A. Results from a) single heuristics and b) collaborative methods are compared to the best known solutions from the literature.

Single Deterministic Heuristics		
Heuristic	Problems Solved	Extra Bins
FFD	393	1088
DJD	356	1216
DJT	**430**	**451**
ADJD	336	679

(a)

	Collaborative Heuristic Models							
	Problems Solved				Extra Bins			
	min	max	mean	sd	min	max	mean	sd
AIS I (Sim et al., 2013)	554	**559**	556	1.4	**159**	165	162	1.4
Island (Sim and Hart, 2013)	552	**559**	557	1.4	**159**	164	162	1.4
NELLI	559	**559**	559	0	**159**	159	559	0

(b)

used a genetic algorithm to evolve a mapping between a (partial) problem-state and the best deterministic heuristic to use, i.e. it focused on heuristic selection and is an offline heuristic (requires a training phase). We also provide comparison to a recent study by (Burke et al., 2012) that evaluated a Hyper-heuristic on 90 instances taken from $ds3$ and $FaIU$. Finally, all results are compared to the best known solutions from the literature on each problem in order to obtain an absolute measure of quality.

5.1.1 Problem set A

Previous methods for obtaining a collaborative set of heuristics for solving bin-packing problems (Sim and Hart, 2013; Sim et al., 2013) involved a *training phase*, in which an algorithm was trained on a set of problems and performance evaluated on a separate test set. Although NELLI does not have a training phase, for consistency and in order to directly compare results, we adopt the same procedure:

- Problem set A (1370 problems) is split into two equal sized sets (adding every second problem to the test set[6])
- NELLI is run for 500 iterations using the training set as the environment $\mathcal{E}$
- The resulting network obtained at the end of the previous step is then presented with all the problems in the test set (685) and the number of problems solved and bins utilised recorded. No further heuristics are added to the system.

Table 6b directly compares the result obtained by NELLI to previous published work. A further experiment was run using NELLI where $\mathcal{E}$ was set to the full set of 1370 problems in A rather than a reduced set of 685 problems. To obtain a comparison to previous work, the algorithm described in (Sim and Hart, 2013) that utilises an island model to find a set of collaborating heuristics was run using the complete set of 1370 problems. These results are given in Table 6 and confirm that the two systems produce solutions of identical quality on a static data-set. However, as we illustrate in the remainder of the article, NELLI has a number of advantages over previously proposed approaches. Specifically, the system is shown to be scalable; it significantly reduces computation time compared to previous approaches; it is shown to adapt efficiently to unseen problems and rapidly changing environments whilst maintaining a memory of previously encountered problems.

Further analysis is given in Table 7 which shows the number of problem instances solved using the specified number of bins more than the known optimum for the set of 1370 problems. NELLI clearly outperforms the individual human designed deterministic heuristics — many of these perform particularly poorly on certain problem instances. On the other hand, the evolved set of cooperative heuristics retained by NELLI solves 97% of problem instances using no more than 1 extra bin.

[6]This ensures an even split of problem instances for each parameter setting between the training and test sets.

Table 6: The table shows results obtained on the complete set of 1370 problems in B. Results using a) single heuristics and b) collaborative methods are compared to the best known solutions in the literature. The results presented also demonstrate the efficiency of NELLI in sustaining the network using a minimal repertoire of heuristics and problem instances.

Single Deterministic Heuristics		
Heuristic	Problems Solved	Extra Bins
FFD	788	2142
DJD	716	2409
DJT	**863**	**881**
ADJD	686	1352

(a)

	Collaborative Heuristic Models							
	Problems Solved				Extra Bins			
	min	max	mean	sd	min	max	mean	sd
Island (Sim and Hart, 2013)	1120	**1126**	1125	1.1	**308**	316	308	1.4
NELLI	1125	**1126**	1126	0.3	**308**	309	308	0.3

Heuristics Retained NELLI				Problems Retained NELLI			
min	max	mean	SD	min	max	mean	SD
6	8	7.1	0.7	26	57	36.9	6.4

(b)

Table 7: Extra bins (δ) required by NELLI and 4 deterministic heuristics compared to the best known solutions from the literature on the complete set of 1370 benchmark problem instances

	Number of Problems Solved Requiring δ Extra Bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
FFD	788	267	78	83	39	16	18	9	18	4	50
DJD	716	281	119	58	48	36	10	16	23	3	60
DJT	863	331	90	26	30	15	11	2	1	1	0
ADJD	686	368	153	76	38	22	12	9	1	5	0
NELLI	1126	202	26	12	2	2	0	0	0	0	0

Comparsion to other Hyper-heuristic approaches Ross et al. (2003) used an evolutionary algorithm to learn a mapping between the state of a partially solved problem and the heuristic that should be applied at any given time, selecting from the deterministic heuristics described in Table 2. This offline approach requires a training phase using a subset of the data. They applied their method to 890 problems from A: using a training set consisting of a subset of 667 problems, they are able to solve 78.8% of the 223 problems in the unseen test set optimally and 95.4% to within 1 bin of optimal . In comparison, NELLI solves 83.4% of the unseen test set optimally and 96.9% to within 1 bin of optimal.

Burke et al. (2012) use genetic proramming to produce a Hyper-heuristic that generates a new heuristic for solving each of 90 of the problem instances from set A. They report excellent results — a success rate of 93% in finding the best known solutions. However, their approach generates 90 individual heuristics; *each* heuristic is generated following 50,000 iterations of the Hyper-heuristic. i.e. 4.5 million iterations in total. Applied to the same 90 problems, NELLI solves 53% optimally, 92% within 1 bin of optimal and 100% within 2 bins: although these results cannot compete directly with (Burke et al., 2012), they are obtained using only 2 heuristics and at most 1080 heuristic-problem calculations. The results are in line with the defined goal of Hyper-heuristics outlined in Section 2.2, i.e. that Hyper-heuristics should be fast and exhibit good performance across a wide range of problems. As shown in the next section, NELLI has additional advantages, in being adaptive and retaining memory.

5.1.2 Problem Set B

The experimental procedure defined above was repeated using the new and larger problem set B in order to ascertain the systems performance on this new set of problems and to provide a baseline for further experimentation.

The system was executed 30 times with each run conducted over 100,000 iterations using the full set of problems as the environment $\mathcal{E}$ and the default parameters as specified in Table 4. A summary of the

Table 8: Number of bins required on the full set of 3968 problem instances

Heuristic	Total Bins	Extra Bins Than Optimal	Problems Solved Optimally
Optimal	320445	0	3968
FFD	327563	7118	491
DJD	330447	10002	920
DJT	325743	5298	1158
ADJD	323566	3121	1279
NELLI	**322820**	**2375**	**1983**

Table 9: Extra bins (δ) required by NELLI and 4 deterministic heuristics on the new set of 3968 problem instances when compared to the known optimal values

	Number of Problems Solved Requiring δ extra bins										
Heuristic	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$	$\delta = 6$	$\delta = 7$	$\delta = 8$	$\delta = 9$	$\delta \geq 10$
FFD	491	2364	442	208	196	51	22	34	68	19	73
DJD	920	1552	468	248	191	100	92	66	57	34	240
DJT	1158	1936	414	141	85	76	52	35	9	2	60
ADJD	1279	2398	209	38	33	8	2	1	0	0	0
NELLI	1983	1708	201	44	27	5	0	0	0	0	0

results is given in Table 8 which also contrasts the results against those achieved using 4 human designed deterministic heuristics. These results are analysed further in Table 9 which gives the number of problems solved using the specified number of bins greater than the known optimal by each of four deterministic heuristics and NELLI.

Table 9 also demonstrates the relative complexity of the problem instances in B when contrasted to the standard benchmarks in A, with respect to the standard set of deterministic heuristics. For example, on problem set A, FFD was shown to solve 56% of the 1370 problem instances using the known optimal number of bins. In contrast, on problem set B, it only manages to solve 12% optimally. NELLI solves 82% of the problems in A optimally, compared to only 50% of the problem instances in B.

Note that the final evaluation of each of the 30 runs gave exactly the same result in terms of the number of bins required to pack each of the problems in B (although the heuristics and problems sustained in each run differed). One of the runs was selected at random and the results obtained by the final set of heuristics for each instance in B were retained for use in the remaining experiments as a benchmark for the problem set.

5.2 Parameter Tuning

A brief investigation of the impact of three of the main system parameters is conducted to determine their influence and justify the default settings.

5.2.1 Concentration c_{init}

The effect of varying the initial concentration of problems and heuristics is illustrated in Figure 5 which shows the results obtained when NELLI was run 30 times for each of $c_{init} \in \{50, 100, 200, 500, 1000\}$. The system was halted after 100,000 iterations. Each box plot summarises the 30 runs conducted. The vertical axis shows the number of bins more than the *best* result that NELLI achieved on problem set B as described previously and presented in Tables 8 and 9. For $c_{init} < c_{max}/2$, increasing the initial concentration improves performance — the increased *initial* concentration increases the time period that both heuristics and problem instances can be sustained without stimulation, thus increasing the probability of eventually finding a heuristic-problem pairing that is mutually stimulatory. However, as $c_{init} \to c_{max}$, the effect is reversed; newly introduced heuristics dominate due to their larger concentration, potentially suppressing previously established heuristics.

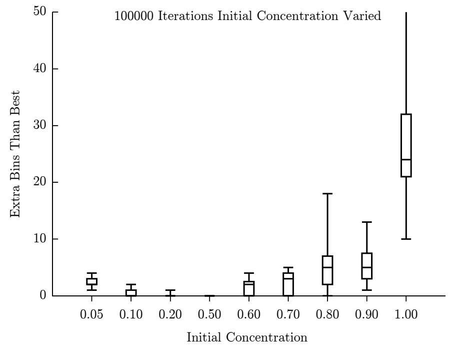


Figure 5: The effect of varying the initial concentration c_{init}. The concentration c_{init} on the x-axis is plotted as fraction of c_{max}

.

5.2.2 Number of problems added per iteration n_p

The parameter n_p describing the number of problems presented to the system each iteration is key in that it has significant impact on the number of calculations that need to be made at each iteration of the algorithm. Each iteration, the number of new calculations C that needs to be performed is given by:

$$C = (n_p \times |\mathcal{H}|) + (n_h \times |\mathcal{P}|) \tag{4}$$

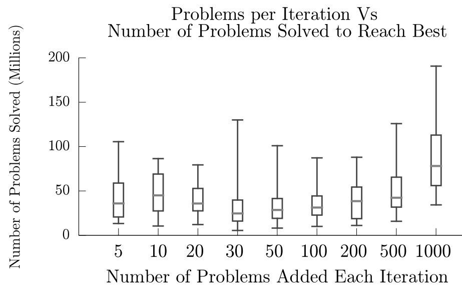
The first term is required to determine the result of applying all heuristics in the system to the new problems just introduced. The second term determines the results of any new heuristics introduced this iteration on all problems currently in the system.

To understand the influence of n_p, the model was executed 30 times for each of 6 different values of $n_p \in \{30, 50, 100, 200, 500, 1000\}$. Each iteration, the cumulative number of calculations undertaken is recorded. The model was allowed to run until the results obtained on $\mathcal{E}$ converged to the *best* result known for the system on problem set B. Figure 6a summarises the results obtained over 30 runs for each parameter setting. The figure shows that increasing n_p. i.e. the number of problem instances presented each iteration has an adverse affect, increasing the overall number of calculations required to achieve the same result. The default value of 30 appears a reasonable choice. Figure 6b shows a single run of the algorithm truncated to 20000 calculations.

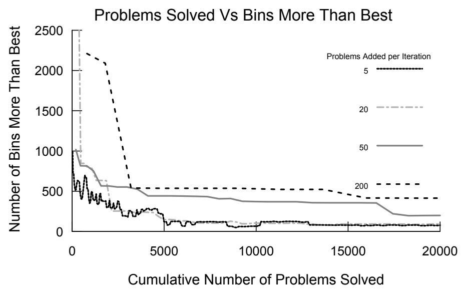
5.2.3 Number of heuristics added per iteration n_h

Figure 7 shows the affect that varying n_h has on the system. For each plot the system was executed for 50000 iterations with $\mathcal{E} = B$ using default parameter settings with the exception of n_h which was fixed for the duration of each plot as shown.

When adding a single heuristic each iteration, a smooth increase in performance is observed over time, and the system converges to the best known result, despite a slow start. Adding a larger number of heuristics per iteration improves the initial performance due to an increased probability of finding good solutions.



(a)



(b)

Figure 6: Number of Problem Instances Added per Iteration Vs Heuristic-Problems Solved to Reach the Best Know Result

However over a longer time scale, performance is hindered, causing undesirable fluctuation in the collective capability of the network. In the worst case, when $n_h = 20$, the system fails to converge to the best result.

As n_h increases, the ability of individual heuristics to find niche areas of the problem space becomes more difficult due to increased competition; newly introduced heuristics are unlikely to gain any stimulation due to the decreased probability of the heuristic solving a problem better than any other heuristic resulting in very short life-times for each heuristic and thus more unstable behaviour in the system. From a computational perspective, increasing both n_p and n_h also significantly increases the number of calculations required each iteration. This further justifies the choice of $n_h = 1$ as the default value.

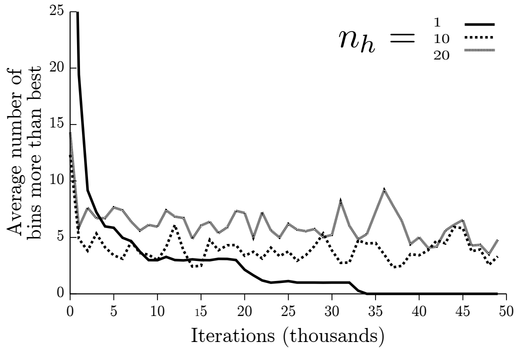


Figure 7: Effect of varying n_h. Results shown averaged on $\mathcal{E}$ every 1000 iterations

5.3 Efficiency and Scalability

To determine the scalability of NELLI (in terms of number of problems in $\mathcal{P}$ with respect to $|\mathcal{E}|$ and $|\mathcal{H}|$ an experiment was conducted in which $|\mathcal{E}|$ was varied, i.e. $|\mathcal{E}| \in \{100, 200, 500, 1000, 2000, 3968\}$. In each case, the problems in $\mathcal{E}$ were randomly selected from problem set B, i.e. $\mathcal{U}' = B$. All other parameters were set to the default values, and the system was run for 50000 iterations over 30 runs. Table 10 shows the mean number of problems and heuristics retained following 50000 iterations of the system. The table also shows the ratio $|\mathcal{P}|/|\mathcal{E}|$, i.e. the fraction of the problems in the environment retained in the network, and the ratio $|\mathcal{H}|/|\mathcal{P}|$ as the size of $\mathcal{E}$ increases, to indicate how the system scales.

As expected, as $|\mathcal{E}|$ increases , the number of retained problems and heuristics increases. Note however that the fraction of problems retained in relation to the environment *decreases*. The problems in the environment $\mathcal{E}$ represent a sample of problems from the larger $\mathcal{U}' = B$. As $|\mathcal{E}|$ increases, more of $\mathcal{U}'$ is sampled, and

	$\lvert\mathcal{E}\rvert = 100$	$\lvert\mathcal{E}\rvert = 200$	$\lvert\mathcal{E}\rvert = 500$	$\lvert\mathcal{E}\rvert = 1000$	$\lvert\mathcal{E}\rvert = 2000$	$\lvert\mathcal{E}\rvert = 3968$
Mean heuristics retained $\mathcal{H}$	5.40	6.87	9.90	12.40	16.83	21.57
Mean problems retained $\mathcal{P}$	18.73	23.3	33.45	41.5	47.4	59.52
Ratio $\mathcal{P}/\mathcal{E}$	18.73	11.65	6.69	4.15	2.37	1.50
Ratio $\mathcal{P}/\mathcal{H}$	0.29	0.29	0.30	0.30	0.35	0.36

Table 10: The table shows the number of heuristics and problems retained in the network as the size of the environment $\mathcal{E}$ increases. All figures obtained over 30 runs and 50000 iterations

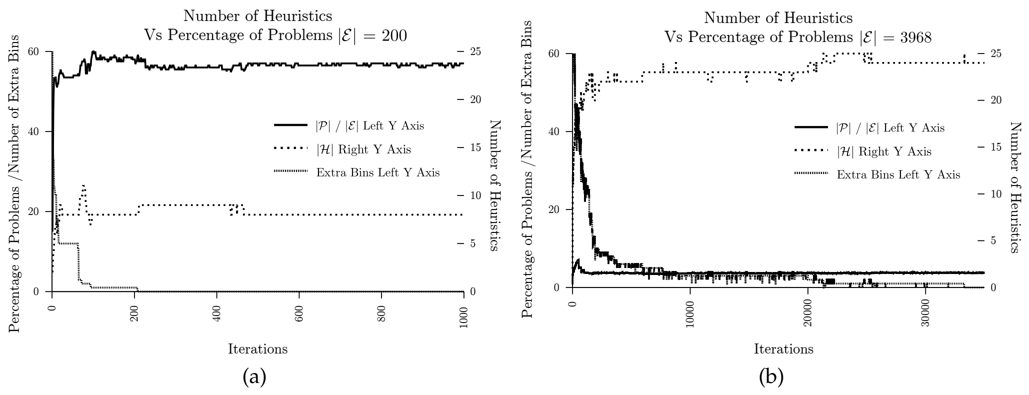


Figure 8: Typical runs for $\lvert\mathcal{E}\rvert = \{200(a), 3968(b)\}$

thus the system is better able to learn a general representation of $\mathcal{U}'$, hence decreasing the ratio of problems $\lvert\mathcal{P}\rvert/\lvert\mathcal{E}\rvert$ required to represent it. This is also reflected in the sub-linear increase in the number of heuristics required as $\lvert\mathcal{E}\rvert$ increases, again confirming the ability of the system to find heuristics that generalise over the environment. The ratio $\lvert\mathcal{H}\rvert/\lvert\mathcal{P}\rvert$ remains almost constant, indicating the scalability of the system. Figure 8 shows a typical run from an experiment for both $\lvert\mathcal{E}\rvert = 200$ and $\lvert\mathcal{E}\rvert = 3968$[7].

With respect to efficiency, we return to the earlier comment made that NELLI is computationally more efficient that its precursors. In the system described in Sim et al. (2013), the complete set of problems in the environment must be evaluated each iteration (i.e. in Equation 4, the final term $\mathcal{P}$ would be replaced with $\mathcal{E}$). In contrast, using NELLI only the sustained subset of problems $\mathcal{P}$ are evaluated. As is clearly shown in Table 10, $\mathcal{P} << \mathcal{E}$ for a range of values of $\mathcal{E}$ — in an environment containing 3968 problems, only 1.5% of these are sustained, hence dramatically reducing computational complexity. Note that to obtain a solution to a new problem instance, it is necessary to apply a greedy procedure in which the performance of each of the $\mathcal{H}$ deterministic heuristics in the system must be evaluted on the instance. Given that for $\mathcal{E} \approx 4000$, the system retains only $\mathcal{H} \approx 21$ heuristics, this does not appear to a limiting factor.

5.4 Continuous Learning Capabilities

In order to demonstrate that NELLI functions effectively as a continuous learning system, it must be tested in a dynamically changing problem environment, in order to demonstrate that is is responsive to new problem and exhibits the plasticity required for network to adapt.

[7]As both heuristics and problems are continually added with sufficient concentration to allow them to survive for at least 3 iterations, then at any iteration, there will be potentially be at most 3 heuristics and 90 problem instances that give no added benefit to the system. The table shows only H and P after the run finishes where any unstimulated problems and heuristics are removed thus the discrepancy between the mean for E=200 being 11.65% in the table and 60% in the figure

5.4.1 Memory and Plasticity: Response to new problems from a similar dataset

Consider the case in which $\mathcal{U}' = B$, i.e. the set of 3968 novel problem instances. At $t = 0$, $\mathcal{E}$ consists of a set of $|\mathcal{E}|$ problems drawn randomly from $\mathcal{U}'$. Every 1000 iterations, $\mathcal{E}$ is replaced with a new random set of problems from B. Experiments are performed in which $|\mathcal{E}| \in \{100, 500, 1000\}$; at each iteration, the size of $\mathcal{H}$ and $\mathcal{P}$ are recorded. Additionally, in order to demonstrate that the system has memory, the performance of the system against every potential problem in $\mathcal{U}'$ is tracked at each iteration. Particularly during early iterations, many of the problems in $\mathcal{U}'$ will not have been presented to the network therefore by measuring the hypothetical response against $\mathcal{U}'$, it is possible to gauge whether the system is generalising from seen instances and retaining that information. As $t \to \infty$, $\mathcal{E}^* \to \mathcal{U}'$.

The results are illustrated in Figure 9. Results are plotted both at every iteration (left-hand column) and averaged over each of the 1000 iterations the problems are present in $\mathcal{E}$ for. A number of trends are clear:

- The network is clearly plastic both in terms of the number of problems and the number of heuristics that are sustained in the network
- NELLI can generalise over $\mathcal{U}'$; even in the early iterations we see good performance across the entirety of $\mathcal{U}'$ when only a small fraction $\mathcal{E}^*$ of it has been presented to the system.
- NELLI continuously learns; the performance measured against all problems in $\mathcal{U}'$ improves over time; the rate of learning can be increased by increasing the size of $\mathcal{E}$, the set of problems currently visible to the network
- NELLI sustains a useful network over time; performance never deteriorates in our experiments providing the parameters are set correctly; the system therefore exhibits memory.
- Increasing $|\mathcal{E}|$, the number of problems in the environment, causes more difficulty at the start but has the effect of increasing the rate of learning overall. This is illustrated further in Figure 10 which summarise the results over 30 runs.

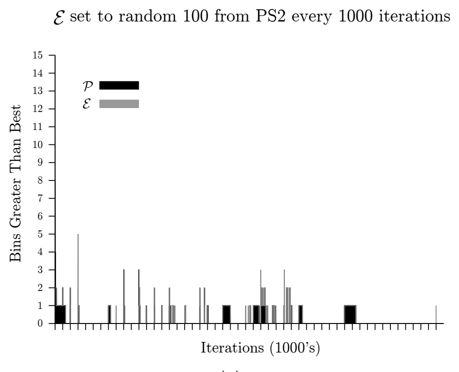
5.4.2 Memory and Plasticity: Response to new problems from different datasets

In order to demonstrate the systems learning and memory capabilities when faced with an environment in which problem characteristics vary over time, experiments are conducted using problems from B_{ds_1} and B_{ds_2}. These data sets — generated from parameters defined by Scholl et al. (1997) are well known to have radically different properties. It is therefore unlikely that a heuristic that performs well on ds_1 will generalise to ds_2.

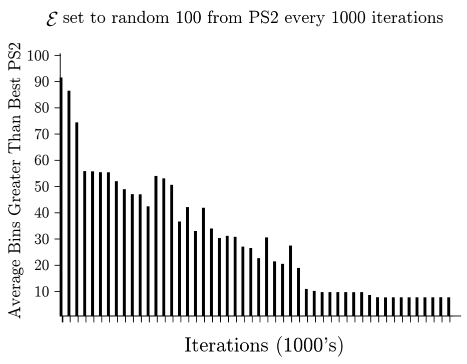
In the following experiments, the environment $\mathcal{E}$ is toggled alternately between B_{ds_1} and B_{ds_2} every 500 iterations. Two experiments were performed:

- The system was restarted every 500 iterations to obtain a benchmark response for the current set of problems presented (equivalent to a system with no memory)
- The problems in $\mathcal{E}$ were replaced every 500 iterations, but the heuristics present were retained (in order to test whether the system retains a useful memory)

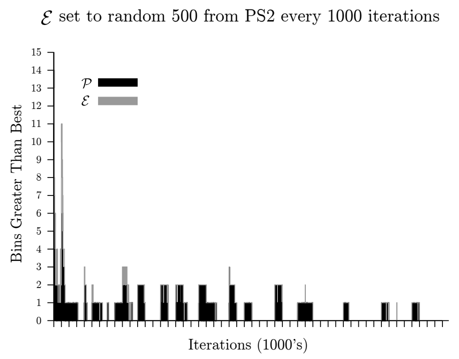
In each of the two scenarios, i.e. with and without memory, we calculate the number of extra bins required to solve problems with respect to the best known solution using the heuristics present in the network every iteration. Results are given in Figure 11 which show the results over a single typical run. In these diagrams, the blocks alternate every 500 iterations to highlight the data-set being considered. All figures are obtained from the same typical run. Figures 11c and 11d show the same information as Figures 11a and 11b but on a smaller scale. Figures 11e and 11f average the results over each 500 iteration cycle. The right-hand column is of most interest, as this shows the metric evaluated over $\mathcal{E}$, i.e. the set of problems in the system environment that we are currently interested in solving. The left-hand column of results



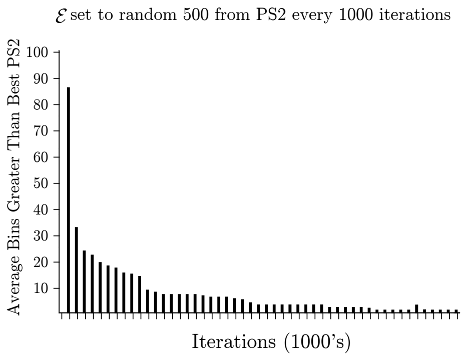
(a)



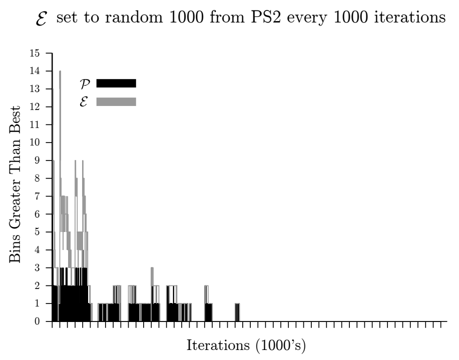
(b)



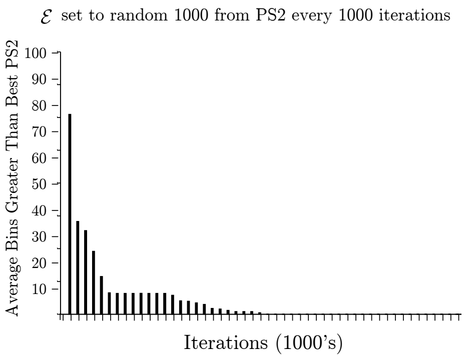
(c)



(d)



(e)



(f)

Figure 9: $\mathcal{E}$ changed every 1000 iterations to a random 100, 500 or 1000 problems from B

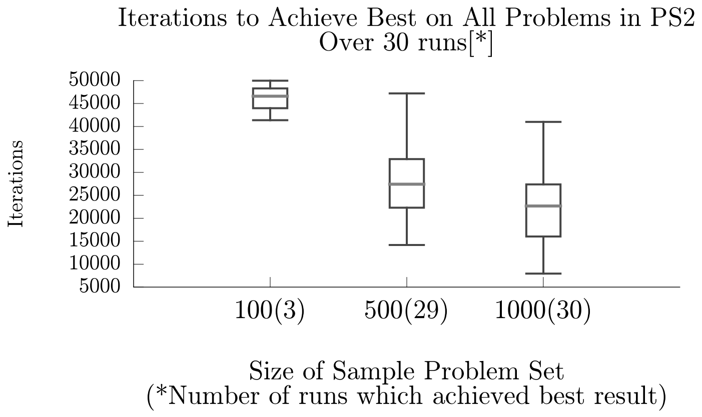


Figure 10: Number of Iterations to reach the best result for different sizes of $\mathcal{E}$

represents the same metric but evaluated over $\mathcal{P}$, i.e. the set of problems that are sustained by the network as being representative of the problem space, and is shown to illustrate how the network is capable of generalising over $\mathcal{E}$ from the problems in $\mathcal{P}$.

We observe the following with regard to $\mathcal{E}$:

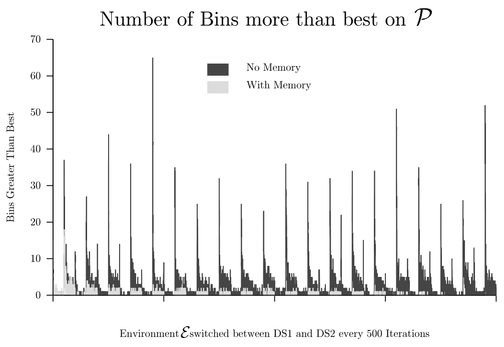
- NELLI — with its implicit memory — always outperforms the system with no memory. Due to the retained network, the system does not have to adapt from scratch to a new environment
- Adaptation still occurs in the system with memory, demonstrating the plasticity of the network
- The memory of a data-set is sustained across cycles in which no items from that data-set are presented. This is apparent in the increasing performance on both data-sets over time.
- B_{ds_1} is clearly much easier than B_{ds_2}: within three presentations of samples from this data-set NELLI has reached optimal performance (i.e 0 bins greater than best) and sustains this performance[8].

Comparing the figures in the right-hand column to those on the left that represent the same metric evaluated over $\mathcal{P}$, we see that performance on $\mathcal{P}$ mirrors that of $\mathcal{E}$, i.e. an improvement on $\mathcal{P}$ correlates to an improvement in $\mathcal{E}$, confirming the generalisation capabilities of the network.

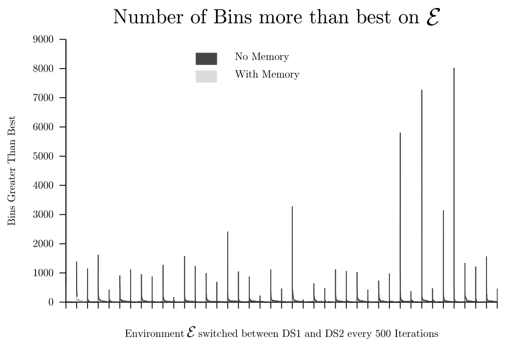
6 Conclusions & Future Work

We have described a continuous learning system inspired by previous work in the Artificial Immune System field that is capable of learning to solve a combinatorial optimisation problem, improves its performance over time and adapts to changing environments. The system fuses methods from SNGP which is used to generate novel heuristics with ideas from immune-network theory, resulting in a self-sustaining interacting network of problems and heuristics that is capable of adapting over time as new knowledge is presented or if the environment changes. When compared to existing approaches (Sim and Hart, 2013; Sim et al., 2013; Ross et al., 2003; Burke et al., 2006) that attempt to find sets of collaborative heuristics, the system performs equally in terms of performance on static data-sets. However it is shown to have significant advantages in its ability to deal with dynamic data; its ability to provide a representative map of the problem space and its computational efficiency. Comparisons to the known optimal results on the suite of 5338 instances tested also show the promise of the system. The test-suite included 3968 problems which were generated in order to provide a harder test than posed by existing benchmark problems; these problems are

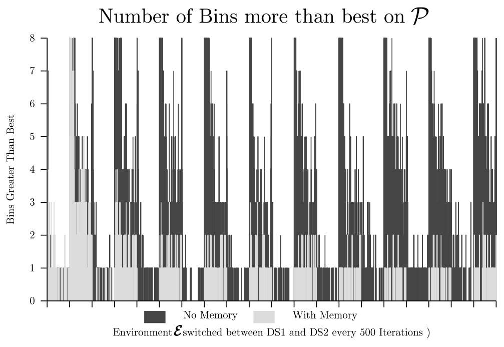
[8]Note that experiments showed that the order in which the two datasets are presented does not have any impact on the results.



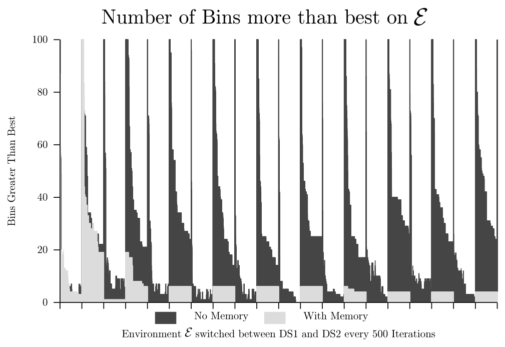
(a)



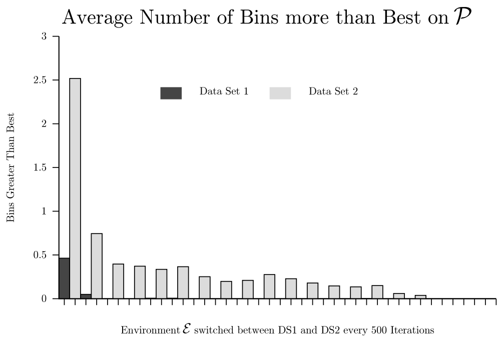
(b)



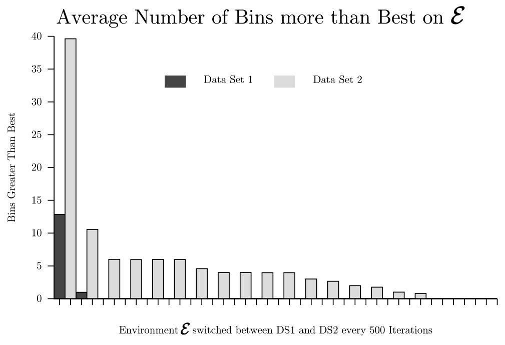
(c)



(d)



(e)



(f)

Figure 11: Alternating $\mathcal{E}$ between B_{ds1} and B_{ds2}. Utility measured against both $\mathcal{P}$ and $\mathcal{E}$

shown to be considerably more difficult than the standard benchmarks and are available as a resource for use by other researchers.

NELLI meets the requirements defined by Silver et al. (2013) for a life-long machine learning system: it incorporates a long-term memory; it can selectively transfer prior knowledge when learning new tasks; it adopts a systems approach that ensures the effective and efficient interaction of the elements of the system. Further, as specified by Silver et al. (2013), it is computationally efficient when storing learned knowledge in long-term memory and retains its knowledge online. The system is shown experimentally to be scalable in terms of the number of heuristics and problems it sustains as the size of the environment increases.

Although the system is tested using 1D bin-packing as an example domain, we believe it will generalise easily to other combinatorial optimisation problems. The underlying principal behind NELLI is that heuristics that are successful are sustained by the network. We generate constructive heuristics using SNGP to combine nodes that are explicitly designed to place one or more items into a solution. There is no requirement to limit NELLI to these types of heuristics or to generate those heuristics using SNGP. Heuristics could be hand crafted or automatically generated using other methods. Future research could consider additional heuristic generation techniques such as in (Burke et al., 2012). Similarly the heuristics used do not have to be limited to deterministic constructive heuristics but could include improvement heuristics and stochastic methods. The main requirements of the system are that a number of heuristics can be used to solve problems from across the domain, and that potential heuristics can easily be represented (and therefore generated) using, for example the SNGP format. Recent examples of using Genetic Programming to evolve novel heuristics in the timetabling (Bader et al., 2009) and 2D stock-cutting (Burke et al., 2010b) domains suggest that this is likely to be the case and provide promising avenues for future development.

Acknowledgements

The authors are grateful to Prof. Peter Ross for comments and suggestions on draft versions of this paper which greatly improved it.

References

Bader, M., Poli, R., and Fatima, S. (2009). Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1(3).

Bader-El-Den, M. and Poli, R. (2008). Generating sat local-search heuristics using a gp hyper-heuristic framework. In Monmarch, N., Talbi, E.-G., Collet, P., Schoenauer, M., and Lutton, E., editors, *Artificial Evolution*, volume 4926 of *Lecture Notes in Computer Science*, pages 37–49. Springer Berlin Heidelberg.

Bersini, H. (1999). The endogenous double plasticity of the immune network and the inspiration to be drawn for engineering artifacts. In Dasgupta, D., editor, *Artificial Immune Systems and Their Applications*, pages 22–44. Springer Berlin Heidelberg.

Burke, E., Hyde, M., and Kendall, G. (2006). Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 860–869. Springer Berlin / Heidelberg.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S. (2003). Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, chapter 16, pages 457–474. Kluwer.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010a). A classification of hyper-heuristic approaches. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer US.

Burke, E. K., Hyde, M., Kendall, G., and Woodward, J. (2007a). Scalability of evolved on line bin packing heuristics. In *IEEE Congress on Evolutionary Computation (CEC07)*, pages 2530–2537.

Burke, E. K., Hyde, M., Kendall, G., and Woodward, J. (2010b). A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942 –958.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In Kacprzyk, J. and Jain, L. C., editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer Berlin Heidelberg.

Burke, E. K., Hyde, M. R., Kendall, G., and Woodward, J. (2007b). Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, GECCO '07, pages 1559–1565, New York, NY, USA. ACM.

Burke, E. K., Hyde, M. R., Kendall, G., and Woodward, J. (2012). Automating the packing heuristic design process with genetic programming. *Evolutionary Computation*, 20(1):63–89.

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E. R., and Mitchell, T. M. (2010). Toward an architecture for never-ending language learning. In *AAAI Conference on Artificial Intelligence*.

Csirik, J., Johnson, D. S., Kenyon, C., Shor, P. W., and Weber, R. R. (1999). A self organizing bin packing heuristic. In *Selected papers from the International Workshop on Algorithm Engineering and Experimentation*, ALENEX '99, pages 246–265, London, UK, UK. Springer-Verlag.

Djang, P. A. and Finch, P. R. (1998). Solving one dimensional bin packing problems. *Journal of Heuristics*.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30.

Farmer, J. D., Packard, N. H., and Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena*, 2(1-3):187–204.

F.O. de Frana, F.J. Von Zuben, L. d. C. (2005). An artificial immune network for multimodal function optimization on dynamic environments. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 289–296. ACM.

Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability : a guide to the theory of NP-completeness*. A Series of books in the mathematical sciences. W.H. Freeman, San Francisco.

Gaspar, A. and Collard, P. (2000). Two models of immunization for time dependent optimization. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 1, pages 113–118 vol.1.

Gent, I. P. (1998). Heuristic solution of open bin packing problems. *Journal of Heuristics*, 3(4):299–304.

Hart, E. (2006). Analysis of a growth model for idiotypic networks. In Bersini, H. and Carneiro, J., editors, *Artificial Immune Systems*, volume 4163 of *Lecture Notes in Computer Science*, pages 66–80. Springer Berlin / Heidelberg.

Jackson, D. (2012a). A new, node-focused model for genetic programming. In Moraglio, A., Silva, S., Krawiec, K., Machado, P., and Cotta, C., editors, *Genetic Programming*, volume 7244 of *Lecture Notes in Computer Science*, pages 49–60. Springer Berlin Heidelberg.

Jackson, D. (2012b). Single node genetic programming on problems with side effects. In Coello, C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M., editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7491 of *Lecture Notes in Computer Science*, pages 327–336. Springer Berlin Heidelberg.

Jerne, N. K. (1974). Towards a network theory of the immune system. *Ann Immunol (Paris)*, 125C(1-2):373–89.

Kira, Z. and Schultz, A. (2006). Continuous and embedded learning for multi-agent systems. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3184–3190.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Kromer, P., Platos, J., and Snasel, V. (2012). Practical results of artificial immune systems for combinatorial optimization problems. In *Nature and Biologically Inspired Computing (NaBIC), 2012 Fourth World Congress on*, pages 194–199.

Louis, S. and McDonnell, J. (2004). Learning with case-injected genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 8(4):316–328.

Nanas, N. and de Roeck, A. (2007). Multimodal dynamic optimization: from evolutionary algorithms to artificial immune systems. In *Proceedings of the 6th international conference on Artificial immune systems*, ICARIS'07, pages 13–24, Berlin, Heidelberg. Springer-Verlag.

Nasraoui, O., Uribe, C., Coronel, C., and Gonzalez, F. (2003). Tecno-streams: tracking evolving clusters in noisy data streams with a scalable immune system learning model. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 235–242.

Pappa, G., Ochoa, G., Hyde, M., Freitas, A., Woodward, J., and Swan, J. (2013). Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms. *Genetic Programming and Evolvable Machines*, pages 1–33.

Ross, P. (2005). Hyper-heuristics. In Burke, Edmund K. Kendall, G., editor, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 529–556. Springer-Verlag.

Ross, P., Marn-Blzquez, J., Schulenburg, S., and Hart, E. (2003). Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics. In Cant-Paz, E., Foster, J., Deb, K., Davis, L., Roy, R., OReilly, U.-M., Beyer, H.-G., Standish, R., Kendall, G., Wilson, S., Harman, M., Wegener, J., Dasgupta, D., Potter, M., Schultz, A., Dowsland, K., Jonoska, N., and Miller, J., editors, *Genetic and Evolutionary Computation GECCO 2003*, volume 2724 of *Lecture Notes in Computer Science*, pages 1295–1306. Springer Berlin Heidelberg.

Ross, P., Schulenburg, S., Marín-Blázquez, J. G., and Hart, E. (2002). Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 942–948, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Ruvolo, P. and Eaton, E. (2013). Ella: An efficient lifelong learning algorithm. *Journal of Machine Learning Research*, 28(1):507–515.

Scholl, A., Klein, R., and Jürgens, C. (1997). Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627–645.

Schwerin, P. and Wäscher, G. (1997). The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp. *International Transactions in Operational Research*, 4(5-6):377–389.

Silver, D., Yang, Q., and Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms. In *AAAI Spring Symposium Series*.

Sim, K. (2013). Bin-packing generator. `http://www.soc.napier.ac.uk/~cs378/bpp/`.

Sim, K. and Hart, E. (2013). Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In *Proceedings of GECCO 2013*, New York, NY, USA. ACM.

Sim, K., Hart, E., and Paechter, B. (2012). A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In Coello, C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., and Pavone, M., editors, *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 348–357. Springer Berlin Heidelberg.

Sim, K., Hart, E., and Paechter, B. (2013). Learning to solve bin packing problems with an immune inspired hyper-heuristic. In *Proceedings of ECAL 2013, 12th European Conference on ALife*. MIT Press.

Terashima-Marín, H., Ross, P., Farías-Zárate, C., López-Camacho, E., and Valenzuela-Rendón, M. (2010). Generalized hyper-heuristics for solving 2d regular and irregular packing problems. *Annals of Operations Research*, 179:369–392.

Thrun, S. and Pratt, L. (1997). *Learning to Learn*. Kluwer Academic Publishers, Boston, MA.

Timmis, J. (2007). Artificial immune systems—today and tomorrow. *Natural Computing*, 6(1):1–18.

Timmis, J. and Neal, M. (2001). A resource limited artificial immune system for data analysis. *Knowledge-Based Systems*, 14(34):121 – 130.

Timmis, J., Neal, M., and Hunt, J. (2000). An artificial immune system for data analysis. *Biosystems*, 55(13):143 – 150.

Trojanowski, K. and Wierzchon, S. T. (2009). Immune-based algorithms for dynamic optimization. *Information Sciences*, 179(10):1495 – 1515.

Varela, F., Coutinho, A., Dupire, B., and Vaz, N. N. (1988). Cognitive networks: immune, neural and otherwise. *Theoretical immunology*, 2:359–375.

Watanabe, Y., Ishiguro, A., and Uchikawa, Y. (1998). Decentralized behavior arbitration mechanism for autonomous mobile robot using immune network. In DasGupta, D., editor, *Artficial Immune Systems and Their Applications*, pages 187 – 209. Springer-Verlag New York, Inc.

Whitbrook, A., Aickelin, U., and Garibaldi, J. (2007). Idiotypic immune networks in mobile-robot control. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(6):1581 –1598.

Whitbrook, A., Aickelin, U., and Garibaldi, J. (2008). An idiotypic immune network as a short-term learning architecture for mobile robots. In Bentley, P., Lee, D., and Jung, S., editors, *Artificial Immune Systems*, volume 5132 of *Lecture Notes in Computer Science*, pages 266–278. Springer Berlin / Heidelberg.

Whitbrook, A. M., Aickelin, U., and Garibaldi, J. M. (2010). Two-timescale learning using idiotypic behaviour mediation for a navigating mobile robot. *Journal of Applied Soft Computing*, 10(3):876–887.