An Improved Immune Inspired Hyper-Heuristic for Combinatorial Optimisation Problems

Kevin Sim and Emma Hart

Institute for Informatics and Digital Innovation
Edinburgh Napier University
Edinburgh, Scotland, UK
k.sim@napier.ac.uk, e.hart@napier.ac.uk

January 2014

The *meta-dynamics* of an immune-inspired optimisation system NELLI are considered. NELLI has previously shown to exhibit good performance across very large set of optimisation problems and to be capable of continuous learning and exploiting memory by sustaining a network of novel heuristics. We address the mechanisms by which new heuristics are defined and subsequently generated. A new representation is defined, and a mutation-based operator inspired by clonal-selection introduced to control the balance between exploration and exploitation in the generation of new network elements. Experiments show significantly improved performance over the existing system in the bin-packing domain. New experiments in the job-scheduling domain further show the generality of the approach.

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. Hyper-heuristic approaches to optimisation have attempted to raise the generality of approaches by focusing on techniques that work well across large sets of problems. However, even these approaches suffer 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 usually applied *tabula rasa*, generating completely novel heuristics (or apply existing heuristics in unique combinations) for every problem solved. This leads to *inefficient* algorithms that fail 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 [17] 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 lifetime.* 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 and 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 [18, 19], the authors noted that the natural immune system exhibits properties that fulfill 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. They describe the implementation a system dubbed NELLI (Network for Life Long Learning) that was applied to a large dynamically changing corpus of 1-d bin-packing optimisation problems with very successful results. The system generated a continuous steam of novel heuristics, which self-organised into a self-sustaining network of heuristics that efficiently collectively covered the problem space using a minimal repertoire of components; and was shown to be plastic enough to adapt to new classes of problem within the bin-packing domain.[1]

In this work we retain the core of the NELLI system, in particular *dynamics* that sustain the network, but focus on improving the *meta-dynamic* element of the system in order to provide a better balance between exploration and exploitation of the heuristic space. A novel heuristic representation is introduced and a method based on clonal selection that improves upon existing heuristics (providing exploitation) while retaining the ability of the system to explore the space by generating novel heuristics.

A brief coverage of the background to this paper is provided before NELLI is described in detail. We then provide details of the novel components introduced, providing results that show a significant improvement over the previous system on the bin-packing corpus. Additionally, we provide new results in the domain of Job Shop Scheduling where a diverse set problem instances are tackled, showing the generality of the approach.

2 Related Work

Although many mechanisms have been proposed to explain immune function, of most relevance to this work is the idiotypic network model proposed by [14]

[1]In this work the set of problem instances supplied to the system changed dynamically over time. The individual problem instances remained static

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 to form a self-sustaining network of collaborating cells, that collectively embodies a memory of previous responses. [3] 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 exactly fit the definition of an LML system.

Idiotypic network theory has been translated into a number of computational algorithms in machine-learning, optimisation and engineering domains (see [21] for an overview). However, very few of these applications really address problems in the kind of complex, dynamic environments envisaged by [3]. [16] describe a system based on an idiotypic network for tracking evolving clusters in noisy data-streams, finding it to be both capable of learning and scalable. However, the majority of relevant work evolving learning and/or memory is found in the robotics domain [23]. Although immune-inspired algorithms are common in the field of optimisation, the majority are applied to solving single instances of problems rather than learning to solve large problem sets. Most also are applied to static problems — an exception is the work of [10, 9] who solve dynamic optimisation problems in which the fitness function varies using an idiotypic network approach in which idiotypic network provides a memory.

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 [4] for a recent and comprehensive overview. However there is little work that *combines* these approaches (see the authors work in [18, 19] for an exception). Furthermore, we are unaware of any work other than our previously published work on NELLI [18, 19] that combines hyper-heuristics with AIS in an optimisation system that learns and adapts over time.

3 Application domains

Two application domains are utilised. The objective of the 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 Job Shop Scheduling Problem (JSSP) requires that J jobs have to be processed on each of M machines in a predefined order that can vary for each job. Each of the M operations making up job j has to be processed on a specified machine m and is denoted by (j_i, m_i). Each operation has a corresponding processing time of p_{jm}. The objective is to minimise the total makespan; the time between starting the first operation and completing the last operation across all machines.

Scheduling and packing problems are amongst the most widely addressed in the hyper-heuristic literature, e.g. [20] and [5] describe recent generative hyper-heuristics for each of these domains respectively. The reader is directed to [4] for an overview of hyper-heuristics.

4 NELLI

Introduced in [18] and described in detail in [19], NELLI comprises 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). Illustrated in Figure 1, NELLI is designed to run continuously; problem instances and heuristics can be added in any quantity at any point. The 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 three components of the original system are illustrated in Figure 1 surrounded by a dotted line. The components outside this boundary represent the improvements to the system which are detailed in subsequent sections.

The dynamics of the system are visualised in Figure 2 In this diagram, the Figure (a) shows a set of problems that have entered the system. In Figure (b) 4 heuristics are added; problems are allocated to the heuristic that solves them best. The problems denoted $P1$ and $P2$ are encompassed by two heuristics, signifying that the heuristics each produce an equal quality solution to those problem instances. H2 is clearly subsumed by H1 and H3. As a result of applying the network dynamics, the situation in (c) results; H2 is removed as it does not add to the collective ability of the set of heuristics. Problems P1 and P2 are removed as the best found solutions to these problem instances are implicitly retained by heuristics H3 and H4 that best solve them.

The central AIS network component of NELLI remains faithful to the original implementation and is described by Algorithm 1 where[2]:

- $\mathcal{U}$ - the complete set of 1370 problem instances.
- $\mathcal{E}$ - the current environment, i.e. the set of problems we are currently interested in solving, i.e. $\mathcal{E} \subset \mathcal{U}$
- $\mathcal{E}^*$ - 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 $\mathcal{N}$

[2]The parameter values used in Algorithm 1 are described later in Section 7

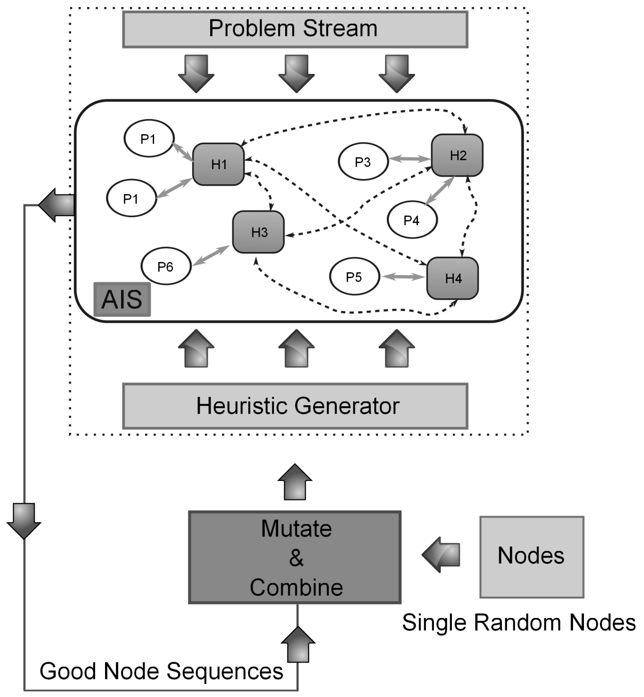

Figure 1: A conceptual view of the system: Problems and heuristics are continuously injected into the AIS. The dynamics and meta-dynamics of the system result in a self-sustaining network of heuristics and problems. Heuristics and problems that receive no stimulation are removed. Solid lines show *direct* interactions between components, dashed lines represent *indirect* interactions. The original system is shown surrounded by a dotted line. The heuristic generator now allows for new heuristics to be created randomly, as before, or by mutating or concatenating heuristic sequences already sustained by the system.

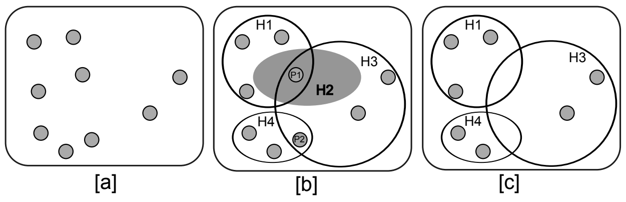

Figure 2: Diagram [a] shows the problems that the system is currently exposed to $\mathcal{E}$. The middle diagram [b] shows a set of generated heuristics that cover the problems in $\mathcal{E}$. The problems $P1$ and $P2$ shown are equally solved by one or more heuristics and therefore not required 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.

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

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

NELLI captures the three essential concepts of an immune network as proposed by [22] – structure, dynamics, and meta-dynamics. The term *structure* refers to the interactions between components of the network, in this case, problems and heuristics as described by steps 5 and 6 of Algorithm 1 and Equations 1 and 2. *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 7-9 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.

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 concen-
        tration $c_{init}$
 5:     calculate $h_{stim} \forall h \in \mathcal{H}$ using Equation 1
 6:     calculate $p_{stim} \forall p \in \mathcal{P}$ using Equation 2
 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
```

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.

Although the implemented dynamics are simplified with respect to much research in AIS that relies on equations defined by [7], the system is shown in [18, 19] to provide heuristics that efficiently cover the problem and collectively minimise the total number of bins used to solve all problems the network is exposed to. 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 there is an *indirect* interaction arising from the method of calculating the problem$\leftrightarrow$heuristic interactions.

5 Novel Improvements to NELLI

This section introduces the improvements made to both the representation used and the processes employed to generate a novel stream of heuristics for incorporation into NELLI.

5.1 Representation

A key feature of NELLI is the ability to generate a constant stream of novel heuristics. In [18, 19], heuristics were defined using a tree representation borrowed from Single Node Genetic Programming (SNGP) [13]. Trees were 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. In this paper we replace this tree representation with a linear sequence of heuristic-components, in which all nodes explicitly cause items to be placed into the solution. Each component, or node, constructs part of a solution, e.g. placing items in a bin in the bin-packing domain, or scheduling an operation in a scheduling domain.

Figure 3 shows a generic example showing a string of five heuristic components with a "pointer" used by an encompassing *wrapper* to indicate the current component position. For each problem instance the general concept for any problem domain is to apply each node in sequence until all objects are placed into the solution. When the pointer reaches the end of the string it is simply returned to the beginning. A wrapper — specific to the problem domain — controls the application of heuristics, testing whether the application of a

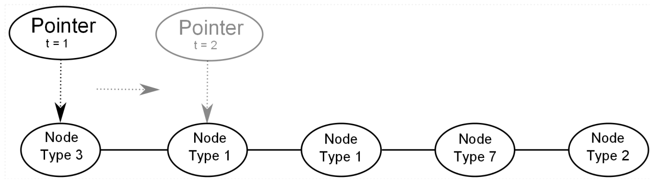

Figure 3: Heuristics are represented as linear sequences of nodes. A pointer keeps track of which node to apply next. The sequence restarts from the beginning after the last node is processed.

heuristic was successful and specifying when to move the next heuristic in the list.

The specific wrappers and set of heuristic components used in the BPP and JSSP domains are described in sections 6.1 and 6.2 respectively.

5.2 Heuristic Generation

Many AIS-based optimisation algorithm employ some form of clonal selection [6] in order to perform exploitation of the search space. Typically, new solutions are generated in numbers proportional to, and with a mutation rate in inverse proportion to the quality of the incumbent. The version of NELLI described in [18, 19] performed only exploration in continuously generating novel heuristics. To balance this, we introduce a simplified clonal selection approach that achieves exploitation but without incurring the overhead that would result from generating and applying a large numbers of cloned heuristics to large sets of problem instances.

A number of mutation operators are used in order to better guide the search for new heuristics towards promising areas of the landscape while maintaining an element of diversity and exploration by still injecting a smaller number of randomly generated heuristics. The dynamics of the system already favours good sequences of nodes, meaning that the number of "good node sequences" retained by the system should be significantly higher than bad combinations. Thus, new heuristics are created by either mutating or concatenating existing sequences currently sustained in the network. Randomly generated heuristics are also injected into the system to maintain an element of diversity and continual exploration of the changing search space. A number of mutation operators are used:

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

- Select two random heuristics from the network and concatenate their nodes to generate a new heuristic.

6 Heuristics and Wrappers

The wrappers and nodes specific to the BPP and JSSP are described in the following sections.

6.1 Bin-Packing

The nodes used are given in table 1 and include only nodes that cause items from the problem to be placed into a solution omitting the function nodes and terminal nodes that encapsulated information about the problem state used in the previous system. A key aspect of the improvements achieved in the bin-packing domain is the way that a sequence of nodes is applied and is described by Algorithm 2.

Algorithm 2 Bin-Packing Problem Wrapper

```
Require: $I \in \{i_1, i_2, ..., i_n\}$ {The set of items to be packed}
Require: $B = \emptyset$ {The set of bins which is initially empty}
Require: $N = (n_1 \ldots n_j)$ {The sequence of nodes}
Require: $p = 1$ {At the start set the pointer to the first node}
  repeat
    add a new bin $b$ to $B$
    repeat
      $I' = I$
      $result = evaluate(n_p)$ {This may cause items from $I$ to be packed into
      the current bin $b$}
      $p = p + 1$
      if $p > j$ then
        $p = 1$
      end if
    until $result < 0$ or $I = \emptyset$ or $I = I'$
    if $I = I'$ and $I \neq \emptyset$ then
      pack each remaining item in a new bin
    end if
  until $I = \emptyset$
```

Unlike the previous application to the BPP the heuristic applied to pack each bin is not identical. 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, the current one is closed and the pointer advances to the next node. The pointer is reset to the start after the evaluation of the last node in the sequence. This results in a different sequence of nodes being applied for each bin increasing the potential utility of the heuristic.

Table 1: BPP 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
B3	Works as for B2 but packs exactly 3 items
B4	Works as for B2 but packs exactly 4 items
B5	Works as for B2 but packs exactly 5 items
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
B4A	As for B2A but considers sets of up to 4 items
B5A	As for B2A but considers sets of up to 5 items

6.2 JSSP

The component heuristics used in the linear sequence for JSSP are defined as follows:

Each of the thirteen rules described in Table 2 can be combined with one of three choices of conflict set described below which is used to determine which operation(s) should be scheduled next:

- **Conflict Set A** Based on Giffler and Thompson's pioneering work[11] gets the operation(s) that can be completed the earliest and selects the machine(s) that those operations are to be processed on. The set of operations returned includes all available operations waiting to be processed on those machines.
- **Constraint Set B** The set of all available waiting operations.
- **Constraint Set C** From [12] returns the subset of available operations that can be started the earliest.

As before, a wrapper advances a pointer through the sequence of nodes starting at the beginning an applying each in turn. Applying each heuristic schedules exactly one operation onto a machine at the earliest possible time. The dispatching rules themselves may return more than one possible operation — in this case one of these is selected randomly and returned to the wrapper to be scheduled.

Table 2: Scheduling Rules

Pseudonym	Packs an operation:
FCFS	Earliest in its job queue
LMT	Longest machine processing time so far
LORPT	Longest remaining job time including the operation
LOS	Longest remaining job time not including the operation
LOT	Longest operation time
LPT	Logest job duration
LRPT	Longest remaining job time not including the operation
RAND	At random
SMT	Shortest machine processing time so far
SORPT	shortest remaining job processing time including the operation
SOT	Shortest op processing time
SPT	Shortest job processing time
SRPT	Shortest remaining job processing time not including the operation

7 Experiments

NELLI as defined by Algorithm 1 requires a number of parameters to be set. The default parameters are detailed in Table 3 and justified in [18]. When applied to the BPP, NELLI was executed using the default parameters settings. On the JSSP the number of problems added each iteration, n_p, was reduced from 30 to 10 to take account of the reduced size of the problem set.

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

Table 3: Default parameter settings for experiments

Two new parameters are introduced. The first, $prob_{mut}$, defines the probability that a new heuristic will be generated by mutating an existing heuristic with the remainder generated randomly. The second parameter, max_{len}, determines the maximum length that a randomly generated sequence of nodes can have (randomly selected between $[1 \ldots max_{len}]$). If mutation is applied to generate a new heuristic then the operator chosen is selected with equal probability. The maximum length of new heuristics generated by mutation is not fixed. The length of the sequence has no affect on the heuristics computational complexity. Nodes are simply evaluated cyclically until all items are placed into a solution.

The experiments conducted were designed to first test the utility of the representation and second the merit of using mutation to exploit good sequences and improve the speed of convergence. The results on the BPP are contrasted against those achieved using the original implementation with those obtained on the JSSP compared to the results obtained by applying the individual rules in isolation. Experiments were conducted on both problem domains using a range of different values of $prob_{mut}$ and max_{len}.

7.1 Bin-Packing Results

For the BPP, 1370 problems instances were used to test the system allowing a comparison to the original implementation where these instances are defined in detail [18, 19]. Results obtained here on the BPP are denoted NELLI* to distinguish them from those obtained previously, denoted for this section as NELLI.

Table 4 gives the results obtained by the best combination of the $prob_{mut}$ and max_{len} parameters. Figure 4 summarises the results obtained using each of the 20 different parameter settings. Each box in the plot shows the result of running NELLI* 30 times with each run terminated after 5000 iterations.

The table clearly highlights the benefit of the approach used by NELLI* in comparison to NELLI, both in terms of the number of problems solved optimally and in the total number of extra bins used across all problems; the number of extra bins is reduced from 308 to 171 and there is an increase in problems solved optimally from 82% to 88.2%. The graph shows the impact of the parameter settings, with a clear trend in improvement shown as the mutation rate is increased, resulting in more exploitation of existing heuristics. Note that even experiments in which the mutation rate was set to zero outperform the previous version of NELLI, indicating the superiority of the newly introduced representation over the previously used tree-structures.

The table shows a trade-off emerging between the number of heuristics sustained and the overall quality — the new exploitation process generates and sustains *additional* heuristics that fill *gaps* in the problem space. An additional experiment limited the maximum number of heuristics allowed in NELLI* to 6 to mimic NELLI — even with this constraint an increase in performance was obtained, of 281 bins over optimal.

Figure 5 expands 5 of the 20 plots (those denoted $010_{0.00} \rightarrow 010_{0.99}$) from Figure 4 plotting them against the results achieved previously using NELLI. It is apparent that NELLI* reacts just as rapidly as NELLI achieving comparable results within the first few iterations but that NELLI* continues to improve if given sufficient time to do so. It is also clear from Figure 5 that increasing the value of $prob_{mut}$ correlates to a more rapid response highlighting the benefit of a more directed search when compared to NELLI and results obtained using NELLI* with $prob_{mut} = 0$. This is reinforced by Figure 4 which highlights an increase in average solution quality with an increase in the probability of generating new heuristics using mutation.

Experiments were also conducted where the *concatenation* and *addition* mutation operators were removed from the system resulting in heuristics that never exceeded their initial length. These results are omitted due to space considerations but highlighted that utility of the concatenation operator in terms of both speed of convergence and final solution quality.

7.2 Job Shop Scheduling Results

A smaller study was carried out on the JSSP to investigate the ability of NELLI to generalise to other problem domains. A total of 62 problem instances were used as a test-bed which were taken from 5 publications, described below.

- 3 instances from Fisher and Thompson [8] referred to as $ft06, ft10$, and $ft20$.
- 40 instances from Lawrence [15]: referred to as $la01 - la40$.
- 5 instances from Adams et al. [1]: referred to as $abz05 - abz09$.

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

	Number of Problems Solved Requiring δ Extra Bins					
	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$
NELLI	1126	202	26	12	2	2
NELLI*	1209	152	8	1	0	0

	Number Heuristics	Number Extra Bins
NELLI	6	308
NELLI*	39	171

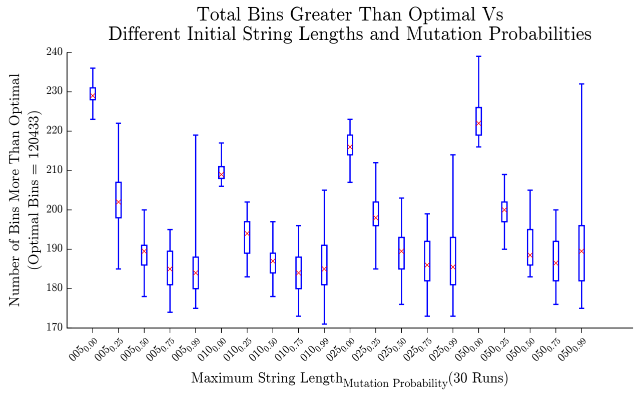

Figure 4: Comparison between different mutation probabilities, $prob_{mut}$, and different maximum string lengths, max_{len}, averaged over 30 runs. The original implementation of NELLI managed to solve these problems using between 308 and 309 extra bins. A t-test comparing the 2 systems (using the results shown here for $010_{0.75}$) show the results to by highly significant giving a P value of less than 0.0001

- 10 instances from Applegate and Cook [2]: referred as $orb01 - orb10$.
- 4 instances from Yamada and Nakano [24]: referred as $yn01 - yn04$.

These problem instances are widely studied by practitioners and have best known solutions, obtained using more specialised metaheuristic techniques, that are superior in many cases to those presented here. Studies using NELLI have to date been limited to the BPP and even here, on the JSSP, the heuristic generator developed is limited to combining simple problem specific nodes with the objective of generating sets of constructive heuristics quickly that can collectively generalise over the problem space. No comparison is made here to other

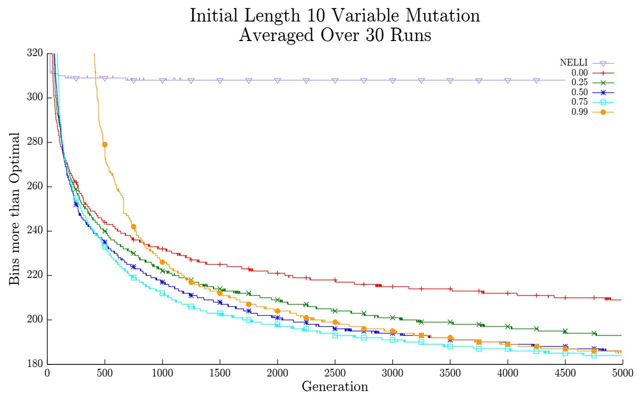

Figure 5: Comparison between NELLI and NELLI* using different mutation probabilities $prob_{mut}$ with an initial string length of $max_{len} = 10$. Results averaged over 30 runs.

work due to the lack of publications addressing the JSSP in the hyper-heuristic literature. Instead the heuristics that emerge are contrasted against results obtained using each of the 39 nodes, described in Section 6.2 when applied in isolation. Results obtained using each of these 39 nodes are presented in Figure 6 which shows for each node the result of applying the rule 30 times to each problem instance.

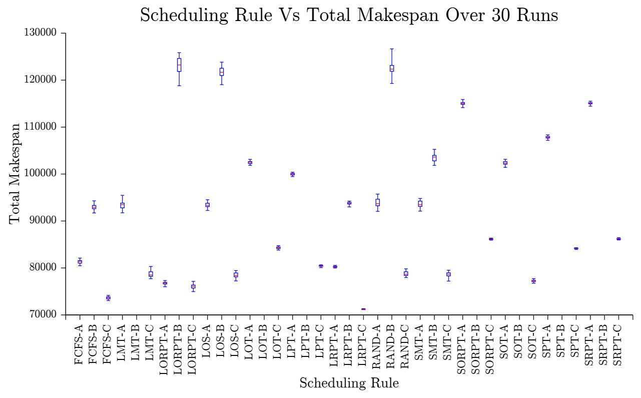

Figure 6: Comparison between scheduling rules applied in isolation taken over 30 runs.

Figure 7 shows the results of repeating the experiments described for bin-packing in which both the mutation rate and maximum string length are varied. The same pattern is observed — increasing the proportion of heuristics that are generated via mutation improves solution quality. In this study there is less of a difference in final solution quality with respect to the initial sequence length although if mutation is not used it is clear that the final solution quality is reduced. Results obtained, and omitted here, using longer initial sequences of dispatching rules showed a marked deterioration in solution quality which it is hypothesised is due to the fact that many more nodes are utilised here than on the BPP resulting in an increased size of the search space. As with the BPP, high mutation rates allow the search to be directed towards more useful sequences of rules although the stochastic nature of the scheduling rules causes new sequences to vary in performance (the same rule can return a different operation if there are still conflicts after they are applied).

The best known reported solutions to these problems have a total makespan summed across all 62 problem instances of 63318. In contrast the best result obtained by a *single* dispatching rule was achieved by LRPT-C which produced solutions with a total makespan of 71130 (12.3% more than the optimal). The best single result achieved using NELLI* gave a combined makespan of 65641 (3.6 % more than optimal).

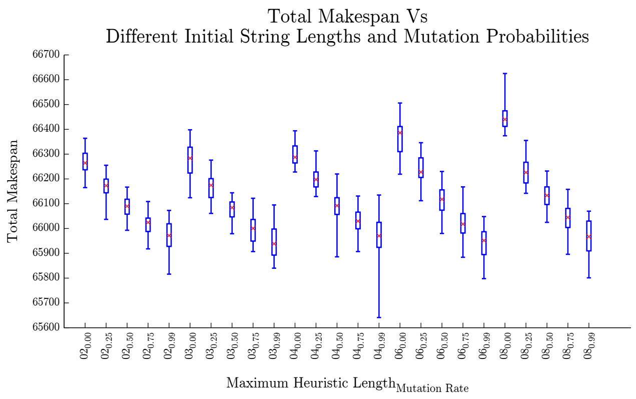

Figure 7: Comparison between different initial maximum string lengths and mutation probabilities taken over 30 runs.

8 Conclusion

An existing immune-inspired system NELLI that enables an optimisation system to solve large sets of problems has been improved by focusing on the *meta-dynamics* of the system, i.e. the mechanism by which the system generates novel heuristics. The original tree-based representation has been replaced with a linear sequence of nodes that is applied cyclically. A mutation mechanism is used to modify heuristics that are currently sustained in the system, acting as an exploitation mechanism. An experimental investigation proved the efficacy of this approach, and investigated the balance required between exploiting existing heuristics and generating completely novel ones. In contrast to previous work, the new representation results in each bin being packed with a (potentially) different heuristic which is hypothesised to contribute to the improvement in quality.

Results on a large set of bin-packing problems showed significant improvement compared both to the previous AIS and existing results in the literature. The system was also applied to a previously untested domain, that of JSSP. A new set of heuristic components and wrapper were defined, but the remainder of the system remains identical to that used for bin-packing. Results showed again that the system outperformed any individual heuristic, and that a balance could be found between exploitation and exploration.

References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

[2] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.

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

[4] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *J Oper Res Soc*, pages –, July 2013.

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

[6] F. M. Burnet. *The clonal selection theory of acquired immunity.* Cambridge University Press, Cambridge, UK, 1959.

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

[8] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice Hall, Englewood Cliffs, New Jersey, 1963.

[9] L. d. C. F.O. de Franca, F.J. Von Zuben. 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, 2005.

[10] A. Gaspar and P. Collard. 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, 2000.

[11] B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503, 1960.

[12] E. Hart and P. Ross. A heuristic combination method for solving job-shop scheduling problems. In A. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature PPSN V*, volume 1498 of *Lecture Notes in Computer Science*, pages 845–854. Springer Berlin / Heidelberg, 1998.

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

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

[15] S. Lawrence. Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh., 1984.

[16] O. Nasraoui, C. Uribe, C. Coronel, and F. Gonzalez. 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, 2003.

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

[18] K. Sim, E. Hart, and B. Paechter. Learning to solve bin packing problems with an immune inspired hyper-heuristic. In G. N. S. N. Pietro Lió, Orazio Miglino and M. Pavone, editors, *Advances in Artificial Life, ECAL 2013: Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems*, pages 856–863. MIT Press, 2013.

[19] K. Sim, E. Hart, and B. Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary Computation Journal*, In Press, January 2014.

[20] J. C. Tay and N. B. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453–473, 2008.

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

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

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

[24] T. Yamada and R. Nakano. Genetic algorithms for job-shop scheduling problems. In *In Proceedings of Modern Heuristic for Decision Support*, pages 474–479. Morgan Kaufmann, 1997.