<table>
<thead>
<tr>
<th>項目</th>
<th>内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>著者（英）</td>
<td>ヴラディミール・ブラゴヴェスト</td>
</tr>
<tr>
<td>学位名</td>
<td>博士工学</td>
</tr>
<tr>
<td>学位授与番号</td>
<td>計算科学甲第 347 号</td>
</tr>
<tr>
<td>学位授与年月日</td>
<td>2007年3月24日</td>
</tr>
</tbody>
</table>

名古屋工業大学学術機関リポジトリ コンピュータ・システム工学領域
HUMAN INSPIRED ALGORITHMS AND ALGORITHM DISCOVERY METHODS FOR ROBOTIC INTELLIGENCE

A dissertation
submitted to the Department of Systems Engineering
of Nagoya Institute of Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Engineering

January 2007
Vladimirov Blagovest Iordanov
Abstract

The objective of the presented research is to identify and implement algorithms providing robotic intelligence. The starting point of the research is implementing algorithms for robotic intelligence using a neural network based cognitive control model, since it has the potential to combine the learning capability of neural networks with the flexibility of symbolic systems. To confirm and utilise this potential, we proposed two algorithms inspired by high-level human cognitive abilities. A further step toward practical application requires improvements of the process for obtaining the algorithms’ inputs. For that purpose, we proposed two algorithm discovery methods targeting algorithms used in low-level, basic information-processing human abilities.

The first proposed algorithm is based on the human ability for mental rehearsal and provides a functionality for fast switching among the learned behaviours according to environment and task requirements. The algorithm is characterised by a neural network implementation of behaviour-based control system that supports learning combinations of basic behaviours, and acquiring internal representations that govern the behaviour switching. Unlike the most approaches that encode the robot’s behaviour only into the neural network’s weights, here it is partially encoded as a sustained activation of the context layer neurons, which allows faster changes. An expected benefit of implementing the proposed algorithm is providing behaviour flexibility for a control system of a cleaning robot or a pet robot, for example.

The second proposed algorithm provides realisation of task specific, human abilities by learning from successful task solutions obtained from skilled individuals. The algorithm is characterised by a modified neural network realisation of a cognitive control model that allows learning from positive examples only. In tests performed using data from human
solved packing task instances, the proposed algorithm compares favourably to other often used algorithms. An expected benefit of the proposed algorithm is to provide robotic intelligence corresponding to the skilled humans’ ability to solve difficult to formalise problems like, for example, scheduling problems.

The first algorithm discovery method creates situations suitable for discovering basic algorithms that support human skill acquisition ability. The method is characterised by the use of implicit rule embedding and transfer observation to allow elucidating which parts of this rule a human subject learns and how. The proposed method resulted in revealing an algorithm used by a human subject to learn the implicit rules hidden behind seemingly random strings in an artificial grammar learning task. The transfer observation helped to distinguish partial rules learned explicitly and such that remain implicit even though the subject reliably uses them to achieve good task performance. An expected benefit of the algorithms discovered with this method is to provide robots with basic skill acquisition abilities.

The second algorithm discovery method provides information about the used input features, rather than the input-output relations. Humans can utilise beneficial environmental features for achieving better task performance, even without being consciously aware of it. We used a simple cognitive experiment to capture this ability to utilise beneficial environmental features. The proposed method is characterised by making explicit which environmental features are used in a given skill, through artificially induced effect of sudden decrease in the task performance (slump) in individuals having this skill. A basic idea for an interface realising the proposed method is presented.

The mental rehearsal ability and the ability to learn from skilled persons’ good problem solutions are representative of our abilities to learn from both, internal and external sources of information relative to ourselves. On the other hand, the ability to acquire implicit rules and the ability to utilise beneficial features are representative of the abilities to learn from internal and external information relative to the frame of the problem, i.e., to learn new features from information, assumed to be pertinent to the problem, and to utilise even information that is normally assumed to be unrelated to the problem.

The mental rehearsal ability and the ability to learn from skilled persons’ good problem solutions are indispensable for realisation of high-level, logical cognitive abilities shown
by highly developed creatures like humans and other primates. On the other hand, the basic algorithms, obtained by the proposed two algorithm discovery methods, are related to primitive cognitive abilities shown even by less developed animals. A parallel use of these algorithms would allow realisation of practically useful robotic intelligence.
Acknowledgements

I would like to thank many people for their support and encouragement during my study.

First and foremost I would like to express my gratitude to my advisor Prof. Hideo Fujimoto for his intellectual guidance and for giving me the opportunity to complete the presented research.

I would like to extend my gratitude to the members of the thesis committee for their invaluable time and advice: Prof. Hiroyuki Ukai for his constructive questions and his enlightening suggestion of the shogi example included in the last chapter, Prof. Akihito Sano for his insightful comments and inspirational ideas, and Assoc. Prof. Hiromi Mochiyama for all the late evening hours spent in helpful discussions, for his guidance, patience, and encouragement in my rougher moments that were crucial for completing this work.

I would like to thank the members and the staff of the Fujimoto Laboratory: Associate Professors Lian-yi Chen, Naoyuki Takesue, and Masamichi Sakaguchi, Research Associates Hirohisa Narita, Ryo Kikuuwe, Fumio Mizuno, Makoto Yagihashi, Yoshihiro Tanaka, and Jumpei Arata for the incredibly diverse and exciting environment.

My heartfelt thanks to Assoc. Prof. Satoko Yamakawa, Atsushi Yamada, Yoshito Ikemata, Yoshitaka Nagano, Takayuki Nakayama, Md. Mohafiquil Kader, Kenta Miyazaki, and Ji Yong for creating a friendly atmosphere and for the stimulating discussions that taught so many things. Thanks to all the students that helped me with the experiments and were always around to explain any difficult word or expression in Japanese.

Finally, I would like to thank my family and friends for their understanding, support, and for being there when I need them the most.
It’s in words that the magic is – Abracadabra, Open Sesame, and the rest – but the magic words in one story aren’t magical in the next. The real magic is to understand which words work, and when, and for what; the trick is to learn the trick.

– John Barth, Chimera –
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>vii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Background</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Objective and Approach</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Organisation of the Thesis</td>
<td>4</td>
</tr>
<tr>
<td>2 Mental Rehearsal for Acquisition of Flexible Behaviours</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Imagine Before You Act</td>
<td>8</td>
</tr>
<tr>
<td>2.1.1 Introduction</td>
<td>8</td>
</tr>
<tr>
<td>2.1.2 Model</td>
<td>9</td>
</tr>
<tr>
<td>2.1.3 Experiment</td>
<td>11</td>
</tr>
<tr>
<td>2.1.4 Discussion</td>
<td>16</td>
</tr>
<tr>
<td>2.2 Preselecting from Alternative Sequences Generated by RNN</td>
<td>17</td>
</tr>
<tr>
<td>2.2.1 Introduction</td>
<td>17</td>
</tr>
<tr>
<td>2.2.2 Problem settings</td>
<td>18</td>
</tr>
<tr>
<td>2.2.3 Method</td>
<td>23</td>
</tr>
<tr>
<td>2.2.4 Experiments</td>
<td>25</td>
</tr>
<tr>
<td>2.2.5 Discussion</td>
<td>30</td>
</tr>
<tr>
<td>2.3 Computational Model and Algorithm of Human Planning</td>
<td>31</td>
</tr>
<tr>
<td>2.3.1 Introduction</td>
<td>31</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>2.3.2 Related Works</td>
</tr>
<tr>
<td></td>
<td>2.3.3 Approach</td>
</tr>
<tr>
<td></td>
<td>2.3.4 Simulation</td>
</tr>
<tr>
<td></td>
<td>2.3.5 Discussion</td>
</tr>
<tr>
<td></td>
<td>2.4 Summary</td>
</tr>
<tr>
<td>3</td>
<td>Learning from Masterpieces</td>
</tr>
<tr>
<td>3.1</td>
<td>Cognitive Control Based Realisation of Human Skill Algorithms</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Problem Description</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Proposed Algorithm</td>
</tr>
<tr>
<td>3.1.4</td>
<td>Comparison Experiment</td>
</tr>
<tr>
<td>3.1.5</td>
<td>Simulation: Generating Problem Solutions</td>
</tr>
<tr>
<td>4</td>
<td>Method for Discovering Algorithms Supporting Human Skill Acquisition Abilities</td>
</tr>
<tr>
<td>4.1</td>
<td>Rule Embedding Approach</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Skill Transfer Observation Based On Rule Embedding</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Artificial Grammar Embedding And Transfer Experiment</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Discussion</td>
</tr>
<tr>
<td>5</td>
<td>Method for Discovering Beneficial Environmental Features Utilised by Skilled Humans</td>
</tr>
<tr>
<td>5.1</td>
<td>A Testing Device for the Human Ability to Utilise Beneficial Environmen tal Features</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Testing Device and Illustrative Example</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Comments on the Idea Behind the Proposed Testing Device</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Discussion</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
</tr>
</tbody>
</table>
# Bibliography

## A Program specifications

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1</td>
<td>Packing Test</td>
<td>112</td>
</tr>
<tr>
<td>A.1.1</td>
<td>Purpose</td>
<td>112</td>
</tr>
<tr>
<td>A.1.2</td>
<td>Implementation</td>
<td>112</td>
</tr>
<tr>
<td>A.1.3</td>
<td>Interface</td>
<td>113</td>
</tr>
<tr>
<td>A.1.4</td>
<td>Settings</td>
<td>113</td>
</tr>
<tr>
<td>A.1.5</td>
<td>Example Session</td>
<td>115</td>
</tr>
<tr>
<td>A.1.6</td>
<td>Output Data</td>
<td>115</td>
</tr>
<tr>
<td>A.2</td>
<td>Bouncing-or-not Test</td>
<td>118</td>
</tr>
<tr>
<td>A.2.1</td>
<td>Purpose</td>
<td>118</td>
</tr>
<tr>
<td>A.2.2</td>
<td>Implementation</td>
<td>118</td>
</tr>
<tr>
<td>A.2.3</td>
<td>Interface</td>
<td>118</td>
</tr>
<tr>
<td>A.2.4</td>
<td>Settings</td>
<td>120</td>
</tr>
<tr>
<td>A.2.5</td>
<td>Example Session</td>
<td>120</td>
</tr>
<tr>
<td>A.2.6</td>
<td>Output Data</td>
<td>120</td>
</tr>
<tr>
<td>A.3</td>
<td>AGL Test</td>
<td>124</td>
</tr>
<tr>
<td>A.3.1</td>
<td>Purpose</td>
<td>124</td>
</tr>
<tr>
<td>A.3.2</td>
<td>Implementation</td>
<td>124</td>
</tr>
<tr>
<td>A.3.3</td>
<td>Interface</td>
<td>124</td>
</tr>
<tr>
<td>A.3.4</td>
<td>Settings</td>
<td>125</td>
</tr>
<tr>
<td>A.3.5</td>
<td>Example Session</td>
<td>126</td>
</tr>
<tr>
<td>A.3.6</td>
<td>Output Data</td>
<td>128</td>
</tr>
</tbody>
</table>
List of Tables

2.1 The input symbols. The second column gives an explanation in terms of the sensory inputs. .................................................. 19
2.2 The output symbols. The second column gives an explanation in terms of the robot’s actions. .................................................. 19
2.3 The four possible sequences. Each sequence consists of five input-output pairs. .............................................................. 20
2.4 Results from 12 test sequences, for the case with fully connected input to hidden layer. The first column shows the contents of the context-plus layer, the next four columns show the how many times the corresponding input-output pair occurs, and the last column gives the number and type of the errors. .............................................................. 27
2.5 Results from 16 test sequences, for the case with tessellated connection from the input to the hidden layer. See the explanation for table 2.4. .................................................. 27
2.6 Test results from using the neural network to control the simulated robot. The second column shows the number of successful runs out of 10, and the last column shows what error occurred. .................................................. 29
3.1 Initial values of the preset weights and the corresponding learning rates. .............................................................. 51
3.2 Groups of items based on the real and assigned classes. .............................................................. 58
3.3 Results of the comparison experiment. .............................................................. 59
4.1 Examples of grammatical and test strings. .............................................................. 73
4.2 Decompositions of the test strings proposed by the skilled subject. The numbers at the split are added later to show the corresponding correct FSA states where possible. .......................................................... 81
4.3 Continuation of Table 4.2. .............................................................. 82
4.4 Some comments made by the skilled subject in the transfer experiment protocol. ................................................................. 83
4.5 Some questions by the naïve subject (Q:) and the corresponding answers by the skilled subject (A:). ........................................ 84
A.1 Initial setting for the packing task program. ................................. 115
A.2 Description of the keys used in the Bouncing-or-not test program. .... 120
A.3 Initial setting for the AGL test program. ...................................... 121
A.4 Description of the keys used in the AGL test program. .................. 125
A.5 Initial setting for the AGL test program. ..................................... 126
A.6 Description of the example test session in Fig. A.6. ......................... 126
List of Figures

1.1 Roomba vacuuming robot. ................................. 2

2.1 An example of mental rehearsal. We are able to imagine how we climb a possible route and prepare in advance for the difficult points. ............... 6

2.2 Extending the available information with memory of past and future states. 7

2.3 Neural network structure ..................................... 10

2.4 An automaton describing the behaviour used to generate the training data. . 11

2.5 An example robot behaviour generated by the automaton description in Fig. 2.4. ........................................... 12

2.6 Hierarchical cluster diagram of the hidden layers’ activity patterns: a) sensory-map, b) s-to-a-map, c) context, and d) action-map; activation based receptive fields for e) smap-sym and f) amap-sym layers .......................... 14

2.7 The robot, its environment, and the four possible action sequences shown with arrows. The large circle in the beginning of the first arrow is the robot. The two small dots after the second arrows are the objects. The area in which the light sources are detectable is shaded. ...................... 20

2.8 A transducer description of the possible sequences. .................... 22

2.9 Structure of the recurrent neural network used in the experiment. ......... 23

2.10 Cluster plot of the hidden layer’s activation patterns for the four sequences, obtained from the network with fully connected input to hidden layer. . . . 26

2.11 Cluster plot of the hidden layer’s activation patterns for the four sequences, obtained from the network with tessellated connection from the input to the hidden layer. ........................................... 28
2.12 The hidden layer’s activation patterns for the four sequences. Rows correspond to the sequences and columns correspond to the steps within a sequence. The activation of a neuron is proportional to the filled area. 29

2.13 a) Reinforcement learning with additional model-generated experience. b) Random walk task settings. c) Neural network structure. 35

2.14 Plot of the average number and standard deviation of left and right sequences over the 10 test trials. The horizontal axis shows the settings for the four simulations in the form: with and without mental rehearsal, each one with the goal on the right and on the left. 37

2.15 Plot of the number of steps used to achieve the specified goal for each training sequence. In this training process, model-generated experience was used, and the goal was shifted every 100 sequences. 38

2.16 Using working memory model for encoding the past and future states. 40

2.17 Adding an imagined experience loop for implementing mental rehearsal. 40

3.1 Example of learning from masterpieces. 41

3.2 Illustration of learning from positive examples only. a) desired solution, b) possible non-optimal solution that can be obtained if starting with random model parameters, c) using an initial bias set to reject all positive examples and gradually expanding the area if the new examples require it. 43

3.3 Outline of the PBWM model. 47

3.4 Structure of the PBWM neural network model, modified for the packing task. 48

3.5 Structure of the hidden and output layers. 49

3.6 Preset weights. Only the first two units of the hidden layer are shown. The rest of the units are identical to the max-value unit. Details of the weights are given in Table 3.1. 50

3.7 Structure of auto-encoding neural network. The input and output layers have equal number of units $d$, while the hidden layer has $m$ units, where $m < d$. 54
3.8 A two-dimensional packing problem is solved by the subjects to obtain training and testing data. Below is shown an example solution. The name, value, and weight are written inside the items. .......................... 57

3.9 Outline of the training and selection processes. .......................... 62

3.10 Plot of the training error. .......................... 64

3.11 Performance index. a) Sum of the size of the packed items. b) Sum of the value of the packed items. .......................... 65

3.12 Plot of the testing errors. .......................... 66

4.1 Outline of the skill transfer observation based on rule embedding strategy. .......................... 72

4.2 FSA describing the grammar (from [11]). .......................... 74

4.3 FSA for generating ungrammatical strings. .......................... 75

4.4 String sets used in the tests. .......................... 76

4.5 Task performance in the artificial grammar embedding phase of the experiment. .......................... 77

5.1 Various skill related devices: a) skillful device, b) coaching device, and c) the proposed testing device, used to identify subjects with skill acquisition potential. (a) and b) are based on [57]) .......................... 86

5.2 The testing device, as seen by the subject, is a computer implementation of an alternative choice task. The subject has to recognise the motion pattern presented on the video display and to respond with a key-press. .......................... 89

5.3 Explanation of the Bounce-or-not task, given to the subject. The two possible motion patterns, shown on the left, are Bouncing and Not Bouncing. A typical test pattern uses similar colours for the discs, as shown on the right. The task is to recognise whether the discs in the test patterns have bounced or not. .......................... 90

5.4 An audio display, placed away from the subject’s view, is added to the testing device. .......................... 91

5.5 In some trials, an additional beep sound is added to the Bounce-or-not task at time $t_c$. .......................... 92

xvi
5.6 Task performance of the subject that passed the test successfully, i.e., was accepted by the testing device. 94
5.7 Task performance of the subjects that failed the test, i.e., were rejected by the testing device. 95
5.8 Performance of the accepted (pass) and the rejected (fail) subjects in the no sound and with sound cases for colour distance $d = 0.02$. 96
5.9 Overcoming asymptotic performance. 99

6.1 A shogi example of humans successfully integrating a - mental rehearsal (e.g., imagine and select future moves), b - learning from good examples (e.g., from books with famous games), c - finding implicit regularities (e.g., specific position patterns), and d - useful environmental features, (e.g., the game partner’s facial expression), to achieve complex intelligent behaviour. 104

A.1 User interface of the program for the packing task. 113
A.2 Example session of the packing task. 116
A.3 User interface of the program for the Bouncing-or-not test. 119
A.4 Example trial of the Bouncing-or-not test - 1a) to 1g). The second trial 2a) begins immediately after the user’s response. 122
A.5 User interface of the program for the AGL test. 125
A.6 Example session of seven trials. Each trial is described by an initial screen with a test string, the user input, a screen showing the program’s feedback, and a trial number given on the left. 127
Chapter 1

Introduction

1.1 Motivation

Recent improvements in the area of robotic hardware lead to increased interest in a wide range of robotic applications including close interaction with humans in normal, everyday life environments. The human interaction aspect of the applications places higher requirements on the robots’ control systems, bringing up the topic of robotic intelligence.

What does it take to achieve robotic intelligence? To discuss this question, we need to clarify what do we understand by intelligence. Following Ross Ashby, we view intelligence as “appropriate selection”:

“... the dual ability to draw appropriate distinctions and to make appropriate, and to a degree better than chance, choices among the things distinguished.”

Then, to focus on robotic intelligence, let us use the Roomba vacuuming robot (see Fig. 1.1) as an example of successfully employed robot in unstructured environments like those found in normal homes. A well-defined purpose like vacuum cleaning allows assessing the appropriateness of the robot’s choices and the necessary distinctions that support them. Furthermore, the purpose is related to one more aspect that we consider important for robotic intelligence – the aspect of usefulness. The iRobot’s claim of over two million

---

2iRobot Corporation, http://www.irobot.com
units sold, suggests that a large enough group of people have found the robot useful, which in turn shows that the robot makes appropriate distinctions and choices. Consequently, we can regard the Roomba robot as a good example of robotic intelligence. The robotic intelligence in the case of Roomba is supported mainly by the behaviour-based control algorithm, which is very well suited for the requirements of the robot’s task.

As this example suggests, robotic intelligence depends not solely on the complexity of the robot’s control algorithms, but also on their appropriateness for the task requirements. In other words, we view the robotic intelligence as a correspondence between the functionality provided by the robot’s control algorithms or mechanical structure and the task requirements.

Based on this notion of robotic intelligence, the presented research is motivated by the idea of enhancing robotic intelligence through identifying and implementing algorithms inspired by human cognitive abilities.

1.2 Background

Many of the early works in the field of artificial intelligence were based on the Physical Symbol System Hypothesis [36], according to which “A physical symbol system has the necessary and sufficient means for general intelligent action”. For example, the General Problem Solver [35], planners as GraphPlan [4] and its modification SGP [55], or cognitive
models as ACT-R [2] were able to obtain good results for problems that can be represented formally using propositional or first order predicate logic. Attempts to extend these results to real-world problems revealed various difficulties, most prominent among which is the frame problem [31]. The symbolic approaches rely on an internal model to find problem solutions, and the frame problem stems from the difficulty to specify what changes and what stays the same after performing an action.

An attempt to avoid this problem by not using an internal model is the subsumption architecture proposed by Brooks [5, 6]. In contrast to the previous approaches, the control problem was decomposed not in terms of functional modules but in terms of task achieving behaviours. It used layered control, tightly coupling perception and action in each added layer. The lowest level layer provided completely functional robot at certain level of competence while the higher levels added new competencies, allowing gradual development. The ideas introduced by the subsumption architecture resulted in shifting the focus from intelligence of the type used in solving logical problems to the type of intelligence that would allow the robots to interact with the real world and successfully perform useful tasks.

Another group of approaches, collectively termed as connectionism [45], attempts to explain and model human intellectual abilities using artificial neural networks. With the supported learning ability, they provide an alternative to the designer-dependent development of the behaviour-based robot control systems mentioned above. With recent improvements in the neural network architectures and learning algorithms [41], such combination may address successfully the need of more complex behaviours while keeping the ability to operate in real-world settings.

1.3 Objective and Approach

We adopt the view of robotic intelligence as a close correspondence between the functionality provided by the robot’s control algorithms and the task requirements. Based on this view, the objective of the thesis is to identify and implement algorithms providing robotic intelligence in the context of application specific, useful functionality. In the search for suitable algorithms, we look for inspiration from human cognitive abilities.
1.4 Organisatio of the Thesis

The thesis is organised as follows.

In Chapter 2, we consider the use of algorithm providing 'mental rehearsal' functionality for robotic intelligence. We develop a neural network realisation of behaviour-based robot control with focus on the flexibility of switching among the learned behaviours. Addressing the problem of acquisition of internal representations and their role in the behaviour switching mechanism, we extend the neural network model to support human inspired, planning algorithm. The implemented algorithm provides 'mental rehearsal' like functionality, resulting in faster switching from one behaviour to another. A viewpoint, from which the obtained plan-like structures that govern the robot's behaviour are treated as strategies, leads to the topic of the next chapter.

In Chapter 3, we consider obtaining successful algorithms from skilled humans for robotic intelligence. A cognitive control based realisation of human skill algorithms is presented. We propose a modification to allow learning from positive examples only. A comparison with two other well used algorithms was performed, using data from human solved packing task instances. In additional simulation, the proposed algorithm was trained and then used to generate packing task solutions similar to the ones used for training. Recognising the need to address more thoroughly the problem of feature selection, we turn our attention basic skill acquisition abilities.

In Chapter 4, we consider the use of basic, fundamental algorithms that support human skill acquisition for robotic intelligence. The proposed algorithm discovery method creates situations suitable for discovering basic algorithms supporting human skill acquisition ability. It was used in an artificial grammar learning and transfer experiment and analysis of the rules acquired by the subject suggested a possible algorithm.

In Chapter 5, we turn to the essential human ability to discover and utilise beneficial environment features. The case in which subjects are able, without being consciously aware of that, to utilise environment features for improving their performance is related to the phenomenon of implicit learning. We present an experiment in which we captured this ability to utilise beneficial environment features.

Finally, Chapter 6 presents conclusions and directions for future work.
Chapter 2

Mental Rehearsal for Acquisition of Flexible Behaviours

Before accomplishing a given task, we are able to imagine a particular way to do it and, also, to imagine ourselves performing the required steps. By doing so, we can identify possible difficulties and prepare for them in advance, or even change our mind and decide to accomplish the task in a different way. An example situation is shown in Fig. 2.1, where a person imagines herself climbing along the chosen route before actually climbing the wall. Such mental rehearsal ability and the particular encoding of the behaviour in the human brain that allow timely changes, support behaviour flexibility. This behaviour flexibility shown by humans stands in stark contrast with the behaviour rigidity of a fly against glass.

In this chapter, we are looking for an algorithm that would provide such mental-rehearsal-like functionality for robots. For this purpose, we need both behaviour encoding that supports easy changes and mechanism for imagining behaviour performance. The underlying idea is to encode part of the robot’s behaviour in the context layer of a recurrent neural network.

The idea is illustrated in Fig. 2.2. Pane a) shows a robot control system that relies only on the present sensory state to decide the current action. This can be achieved, for example, with a feed forward neural network. The upper half of pane a) shows some example cases that can be handled by such system. To take the correct turn at the T-junction, the robot needs an appropriate hint from the environment delivered at that time step. Pane b) shows
Figure 2.1: An example of mental rehearsal. We are able to imagine how we climb a possible route and prepare in advance for the difficult points.
the case when a hint from the environment is presented at a previous time step. To handle this case correctly, the control system needs to keep a memory of previous sensory states. This can be achieved, for example, with a recurrent neural network. Pane c) shows the proposed idea, namely, to extend the memory to include future sensory states too, and to use them to guide the robot’s behaviour. Then, changing the behaviour can be achieved through manipulating these future sensory states.

In the following sections, we first investigate the internal representations formed in the hidden and context layers of the used neural network model. We confirm that the information encoded into the context layer can be interpreted in terms of the robot’s sensory states and actions. Then we confirm the possibility to change the robot’s behaviour through manipulating the contents of the context layer. Finally, we implement and test the algorithm using a working memory model, modified to support learning from imagined experience.

Figure 2.2: Extending the available information with memory of past and future states.
2.1 Imagine Before You Act

2.1.1 Introduction

Considering both operational and behavioural autonomy of a robot, we can distinguish three problems posed to its control system. The first one has to do with acquiring specific behaviours (e.g. approaching, avoiding), learning and adapting them. The second one is coordinating and controlling the interactions among the individual behaviours. And the third one, being a learning problem, concerns the ability to learn different coordination strategies used to solve the second problem [58].

While the specific behaviours and the coordination strategy can be implemented in advance by the designer, often it is a nontrivial task and relies on a priori knowledge about the environment in which the robot operates. This makes a learning capability in the robot desirable. There are many examples in which the learning functionality is provided by the use of neural networks. Among them, we would like to mention three, which focus on the modularity obtained in the learning process.

In Calabretta et al. [7], the authors evolve a modular feed-forward neural network to control a robot, and show that their approach “can lead to a matching between specific behaviours and their structural representation, i.e., to functional modularity”. An approach based on mixture of experts is described in Tani and Nolfi [50]. Each expert module is realized by Recurrent Neural Networks (RNN) and specialises in predicting a distinct part of the sensory flow. In result of this specialisation, a set of representational primitives emerges in the process of learning to predict the next sensory input. The third example is an analysis of RNN made in Ziemke [59], in which the author points out the difference between synchronically and diachronically structured robot control mechanisms. The two example architectures mentioned earlier belong to the first type, i.e. they consist of several different input-output mappings realized as separate modules existing at the same time, and a coordination mechanism to switch among the modules. On the other hand, RNN belong to the second type, since they “integrate several different input-output mappings by dynamically adapting the way they map inputs to outputs in a context-dependent fashion, and thus function as if they were modular in the conventional sense” [59].

The point of view described in the last example, namely, that the RNN are capable of
learning a set of input-output mappings and dynamically switch among them depending on
the internal state accumulated in the context layer is a starting point of the discussion in the
next section. Focusing on the context formation, we consider the possibility to extend it to
include not only a few recently experienced states, but also ‘imagined’ future states. Further
in addressing the problem, we continue in the third section with an analysis of the activity
patterns in the hidden layers of our model, trained with approach-and-grip sequences.

2.1.2 Model

Since the context in RNN plays a significant role in defining the input-output mapping, we
would like to focus on its contents and formation process. In Simple Recurrent Network
(SRN) [17], in addition to the input, hidden, and output layers there is a context layer,
which usually has the same size as the hidden layer and its state is a one-to-one copy of
the hidden layer’s state from the previous step. Furthermore, the context layer may take as
an additional input its own previous state, decayed appropriately. The decaying parameter
influences the depth of the history kept in the context layer.

Meeden trained a robot controlled by a variant of SRN to achieve a particular task, and
then analysed the hidden layer’s contents. She proposed that (under certain constraints)
the activation state of the hidden layer at the moment of achieving a goal “will contain
information that could be used to plan for that goal” [32]. Since these representations in
the hidden layer encode the structure of the robot’s behaviour strategies to some extent, she
calls them protoplans. Even more, Meeden achieved some success in using the protoplans
learned in one controller to provide guidance to a second controller learning the same task.

In the simple case, the hidden layer’s activation state encodes both the current action
and the environment state as perceived through the sensors. This allows the context layer to
keep a trace of the recently experienced states. When for the same sensory input there are
several competing motor outputs, the context layer’s contents serves to bias the competing
hidden layer activations in favour of one of them, and thus to switch from one sensorimotor
mapping to another. Consequently, the decision on which action to take is based on the
history kept in the context. Then, is it possible to keep in the context layer not only a trace
of recently experienced states but also a trace of imagined states, and to allow the robot to
base its choice on the anticipated outcome of its actions? Here follows a brief description of our tentative approach to the problem and comments on some major issues.

![Neural network structure](image)

Figure 2.3: Neural network structure

As a basic model we use a RNN with three hidden layers and a context layer (Fig. 2.3). The input and output layers are defined having in mind a Khepera robot with a gripper, proximity sensors, and a sensor indicating presence of an object in the gripper. The hidden layers are respectively: sensory - \((sensory-map)\), associative - \((s-to-a-map)\), and action - \((action-map)\) areas. The purpose of the \(smap-sym\) and \(amap-sym\) layers will be explained in the next section. Besides the forward flow of information from the sensors through the hidden layers to the motors, there should be another one, back from the associative area toward the sensory area. Its purpose is to bias the sensory area state to reflect the expected next sensory input - reafference process [22, 23]. By varying the influence on the sensory area activation state coming from the expectations and that coming from the actual sensory readings, the robot should be able to ’cut off’ itself from the real world and ‘imagine’ possible future states. In the context layer we would like to keep a trace of imagined future states, somewhat similar to keeping a trace of the hidden layer activations in the SRN. The difference comes from the need to update the context layer selectively according to some criteria, which implies a mechanism to decide which hidden layer activations should be included into the trace. The idea is that the so formed trace of imagined future states in the context layer would bias the consequent robot behaviour in such way as to repeat the imagined sequence. This coarse description brings some rather difficult questions. How to implement the reafference process? How to implement the selective updating of the
context layer, and how to specify the selection criteria? How does the trace of imagined future states guide the consequent behaviour?

Quite straightforward approach to predicting the sensory inputs is described in Jirenhed et al. [29]. Although successful to some extent, their results also show the difficulties in predicting directly the raw sensory readings. This, combined with the idea that a robot, based on its experience, should be able to evolve its own categories through which to perceive the world, prompted us to look for predicting at more abstract level. Our decision to include two additional hidden layers was influenced also by the views, about the role of distributed representations in the hidden layers and their relation to the symbols in the traditional symbolic approaches, presented in Harnad [25] and Chalmers [9].

To provide a base for addressing the problem of next sensory state prediction, we continue with an analysis of the activity patterns in the hidden layers, formed in the learning process.

2.1.3 Experiment

The neural network implementation we used in the experiment is based on the LEABRA framework [38, 41]. It combines error-driven and Hebbian learning, and implements k-Winners-Take-All inhibitory competition that allows for formation of sparse distributed patterns of activity.

Figure 2.4: An automaton describing the behaviour used to generate the training data.

The neural network implementation we used in the experiment is based on the LEABRA framework [38, 41]. It combines error-driven and Hebbian learning, and implements k-Winners-Take-All inhibitory competition that allows for formation of sparse distributed patterns of activity.
Figure 2.5: An example robot behaviour generated by the automaton description in Fig. 2.4.
The diagram in Fig. 2.4 shows the action sequences used to train the network. Although not shown explicitly, each action may be repeated several times before the transition to the next key sensory state presented in the diagram occurs. Generally, the sequences are as follows. The robot wanders around while there is no obstacle in sight. When the proximity sensors signal an obstacle, the robot turns left or right in order to face that obstacle. Then follows an approaching, to distinguish between small and big obstacles (only the small-obstacle case is shown, since only that case was used in the training). Next, the grasping consists of moving back, stopping, moving the arm down, closing the gripper, and finally moving the arm up. In the training, we used two sequences: one with an obstacle appearing on the left, and one with an obstacle on the right. Figure 2.5 shows an example sequence in which the object to be grasped is on the right.

We should note that the focus in this experiment was not on how to generate a grasping behaviour, but on the activity patterns formation in the hidden layers while training the neural network to generate such behaviour. We are not just looking for a neural network model that can learn the desired behaviour, but for a model that develops an abstract description of the behaviour while learning it.

After training, the activity patterns in the hidden layers during one action sequence were subjected to hierarchy clustering based on Euclidean distance. The resulting hierarchy structures are shown on Fig. 2.6, a) - d).

Looking at the clusters formed in the sensory-map layer, we can see that with one notable exception, they correspond to the type of the action taken by the robot. The gss group is divided into two clusters, and one of them is shared among gss and gbs. (The tls patterns that show up into the other cluster with gss are misclassified due to early switching from tls to gss.) This may be explained in the following way, since the other action types do not share same sensory input states, their clusters represent also the distinct sensory states in which the actions are relevant. On the other hand, since the gss and gbs share common sensory input states, they share also the same cluster. The gss group is divided into two clusters that correspond to an obstacle close in front of the robot, and one that is farther. Thus, we may say that the clusters in the sensory-map layer reflect the similarity in the sensory input more than the similarity in the actions.

In the cluster diagram of the s-to-a-map layer, the gss and gbs groups are already in
Figure 2.6: Hierarchical cluster diagram of the hidden layers’ activity patterns: a) sensory-map, b) s-to-a-map, c) context, and d) action-map; activation based receptive fields for e) smap-sym and f) amap-sym layers
different clusters. The history trace kept in the context layer biases toward the one or the other, resulting in the proper action being taken regardless of the identical sensory input. Also, we can mention that the s-to-a-map layer still retains the distinction between an obstacle being close/farther in front of the robot.

Some clusters in the diagram for the context layer have a specific curved shape, clearly seen in the case of long sequence of repeated actions of the same type. It is due to the decay of the pattern corresponding to the previous action type and settling to the pattern corresponding to the current action type. Thus, the misclassification of tls as gss due to early switching results in tls labelled patterns showing at the curved end of the cluster for gss.

Except than being clearly formed, we can mention that the clusters in the action-map layer are divided into two distinct groups, tls, gss, and gbs in the one, and the rest in the other. This may be explained by the fact that tls, gss, and gbs share different combinations of the same output units for the motor control.

In the second part of the experiment, the smap-sym and amap-sym layers were trained, using Hebbian learning, to create localist representations of the clusters in sensory-map and action-map layers respectively. This was prompted by the work of Cangelosi, Greco, and Harnad [8]. For now, the purpose was to analyse the activity based receptive fields of the units in these layers, and thus to see the clusters formed in the sensory-map and action-map layers in terms of activity patterns in the input and output layers. The result is shown on Fig. 2.6, e), f) (e.g. the averaged activity patterns in the input and output layers while unit 0 of the smap-sym layer was active, are shown in the first group labelled un[0]). We can see that the smap-sym units and the corresponding clusters of distributed patterns of activity in the sensory-map layer capture key sensory states in the trained sequences. For example, unit 0 corresponds to an object farther in front of the robot, units 1 and 7 - object close in front, unit 8 - object on the left, unit 6 - arm down, closing the gripper, unit 2 - the object is grasped and the arm is up, etc. On the other hand, the amap-sym units capture the action types: unit 0 is active during tls, unit 9 corresponds to trs, unit 5 - gss, units 1 and 4 - gbs, etc. Note that here gss and gbs have distinct representations.

To summarise, while the formation of activity pattern clusters in the action-map layer seems naturally following the output categories (regarding the sensorimotor mapping as
categorisation problem), the clusters in the *sensory-map* layer reflect a similarity in the sensory input. The cluster formation, besides on the training data, depends also on several parameters internal to the NN model. Among them, we can mention the number of active neurons forming a distributed pattern, the ratio of the pattern size to the layer size, etc.

### 2.1.4 Discussion

Focusing on the role of the RNN context layer in providing modularity in the input-output mapping, we considered the possibility of extending that context to include also ‘imagined’ future states. To get a better understanding of the activations in the hidden layers and the context layer, we presented an analysis of the activity patterns formed after learning two action sequences. This analysis confirmed that in the hidden layers we can obtain meaningful, higher level abstractions from the sensory inputs and the motor outputs. An important point is that we only trained the network to learn the desired mapping from (the low level) inputs and outputs, and did not provide specific training signals to form these abstractions (e.g., this sensory input belongs to the ‘object on the left’ category). These abstractions were autonomously developed by the network as a means to achieve the desired input-output mapping. These results are encouraging for an attempt to predict the next sensory state at more abstract level than the raw sensory readings. Though, (according to the results of the preliminary experiments so far) to succeed in this attempt we need a better control of the pattern formation, so that the patterns in *sensory-map* and *s-to-a-map* retain enough information (i.e. be diverse enough), to facilitate the prediction process. As a next step, we intend to investigate approaches for creating appropriate patterns in the context layer so that we can obtain desired behaviours.
2.2 Preselecting from Alternative Sequences Generated by RNN

2.2.1 Introduction

Due to their learning capability, neural networks have been used at different levels in mobile robot control. While using single network to learn simple behaviour is quite straightforward, attempts with more complex behaviours, requiring diverse and sometimes conflicting input-output mappings, lead to the problem of network modularity. In [7], the neural network modularity is a result of evolutionary process, i.e., the designer does not have to specify which modules are to be used and how. The switching among the modules may be based directly on the current inputs, or in the more difficult case, there is a time interval between the relevant inputs and the switching.

An interesting approach dealing with both modularity and dependence on previous inputs, can be seen in [59]. Ziemke investigates the mechanisms by which some types of recurrent neural networks handle these two problems. The networks are evolved to control a simulated Khepera robot. The environment is divided in two parts by a line drawn on the floor, several identical objects are placed in both parts, and the robot is required to avoid the objects in the one part and to go through the objects in the other. The first-order recurrent neural network controlling the robot solves the task by developing a number of distinct activation patterns in the context layer. These activation patterns, affecting the input-output mappings, allow the robot to respond to similar/same inputs with different actions, e.g. avoiding or approaching an object. Thus, without having explicit modular architecture, the recurrent neural networks develop different input-output mappings and use the context layer activation patterns to influence or change the active mapping.

This role of the context layer leads naturally to the question whether extending its contents beyond the recent history of the hidden layer’s activation states will bring some benefit. An attempt in this direction is presented in this section. We deal with the case when a robot has more than one possible action for a given sensory input and the context layer does not provide enough information to help distinguishing among these actions.

Similar situation arises sometimes due to sensor aliasing or inability of the context layer
to capture and retain the relevant information. Another possible scenario is when the task requirements are subject to changes during execution. One simple approach is to retrain the network when the task changes. A better approach (provided we have the necessary a priori knowledge) is to train for a range of possible tasks and to provide the network with ability to switch (easily) among alternative action sequences based on information available only at run time. This can be done, for example, by adding more inputs, and training the network to associate each of the added inputs with a possible sequence of actions. Later, to switch among the alternative sequences we have to activate the relevant input. However, such implementation depends on external (to the network) mechanism for deciding when and to which sequence to switch.

The approach and the experiment presented here are inspired by the activation based processing discussed in [41, ch.11] and [44]. There, the authors present a model capable of fast switching among a set of tasks. The recurrent neural network, trained to perform a variant of the Wisconsin card sorting task, develops suitable activation patterns in the context layer, facilitating the task switching. A reinforcement learning mechanism controls the setting of an activation state, appropriate for the current task.

Here, we investigate a possible application of this approach to handle a sequence of choices. Our focus is on the hidden layer's activation patterns developed in the process of learning the input-output mapping, since we want to use them to bias the subsequent choices in the desired direction. Also, we look into the interference problems arising from having to use the said patterns to make several choices in a sequence. We analyse the results of the experiment to identify the difficulties with this approach and to look for possible solutions.

2.2.2 Problem settings

To illustrate the problem we need a recurrent neural network trained with sequences of input-output pairs, which have the following properties:

- three inputs having more than one viable output;
- one of these inputs having its outputs dependent on previous inputs/outputs (to make use of the context layer);
the other two inputs having their outputs independent of the previous history.

In this section, we describe the problem in terms of robot action sequences, give a more general description using finite state transducer, and comment on the recurrent neural network used later in the experiment.

**Robot and environment**

We use YAKS (a Khepera robot simulator), with a world containing the robot, a line drawn on the floor, two small round objects, and light sources placed along the outer wall (see Fig. 2.7). The robot sensors are preprocessed, so that we obtain the 4 cases shown in table 2.1. The available actions are described in table 2.2.

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no obstacles near</td>
</tr>
<tr>
<td>1</td>
<td>line detected</td>
</tr>
<tr>
<td>2</td>
<td>small object detected</td>
</tr>
<tr>
<td>3</td>
<td>light detected</td>
</tr>
</tbody>
</table>

Table 2.1: The input symbols. The second column gives an explanation in terms of the sensory inputs.

<table>
<thead>
<tr>
<th>symbol</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>move straight</td>
</tr>
<tr>
<td>a</td>
<td>avoid</td>
</tr>
<tr>
<td>c</td>
<td>cross the line</td>
</tr>
<tr>
<td>g</td>
<td>get the object</td>
</tr>
<tr>
<td>r</td>
<td>release the object</td>
</tr>
</tbody>
</table>

Table 2.2: The output symbols. The second column gives an explanation in terms of the robot’s actions.

The starting position and orientation of the robot are such that moving straight will lead first to encountering the line. Here the robot is presented with the choice whether to cross the line or to avoid it. Regardless of the action taken, continuing with move-straight will lead to a small round object. Again the robot has to choose between grasping the object or avoiding it. Moving straight after that, leads to approaching a light source. At this point,
Table 2.3: The four possible sequences. Each sequence consists of five input-output pairs.

<table>
<thead>
<tr>
<th>Seq 1</th>
<th>Step 0</th>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
<th>Step 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0/m</td>
<td>1/a</td>
<td>0/m</td>
<td>2/a</td>
<td>0/m</td>
<td>3/a</td>
<td></td>
</tr>
<tr>
<td>Seq 2</td>
<td>0/m</td>
<td>1/a</td>
<td>0/m</td>
<td>2/g</td>
<td>0/m</td>
<td>3/r</td>
</tr>
<tr>
<td>Seq 3</td>
<td>0/m</td>
<td>1/c</td>
<td>0/m</td>
<td>2/a</td>
<td>0/m</td>
<td>3/a</td>
</tr>
<tr>
<td>Seq 4</td>
<td>0/m</td>
<td>1/c</td>
<td>0/m</td>
<td>2/g</td>
<td>0/m</td>
<td>3/r</td>
</tr>
</tbody>
</table>

Figure 2.7: The robot, its environment, and the four possible action sequences shown with arrows. The large circle in the beginning of the first arrow is the robot. The two small dots after the second arrows are the objects. The area in which the light sources are detectable is shaded.
the robot has to release the object if it is holding one, otherwise it should avoid the light. Since we do not use the gripper sensor, the robot needs to keep a memory of its previous actions (grasped the object or avoided it). That is how we get the four possible sequences shown with arrows in Fig. 2.7. The same sequences are described with sensory-action pairs in table 2.3.

**Transducer**

While not required in order to obtain the final results, using a transducer to describe the action sequences helps in our case to visualise the problem. It serves also as an abstract description of this class of problems not restricted to the particular realization (i.e., the robot, sensors, and actions we decided to use). Besides, there are research results giving a helpful insight into the relation between FSA and RNN. For instance, Tino [51] shows that a RNN can be trained to simulate an Initial Mealy Machine (i.e. to have the same mapping from input to output strings) and then the automaton can be extracted back from the network by analysing the state units’ activation. In other words, the state unit’s develop activation patterns corresponding to the automaton’s states.

For the robot and the environment described above, considering the sensory inputs as input symbols and actions as output symbols, the input string is only one - “010203”, while the output strings are four - “mamama”, “mamgmr”, “mcmama”, and “mcmgmr”. This transformation is captured by the transducer $M$ shown on Fig. 2.8 and defined as follows.

$$M = \langle Q, \Sigma, \Delta, \delta, q_0, F \rangle$$

for

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$,

$\Sigma = \{0, 1, 2, 3\}$,

$\Delta = \{m, a, c, g, r\}$,

$\delta = \{(q_0, 0, q_1, m), (q_1, 1, q_2, a), (q_1, 1, q_2, c), (q_2, 0, q_3, m), (q_3, 2, q_4, a), (q_3, 2, q_5, g), (q_4, 0, q_6, m), (q_5, 0, q_7, m), (q_6, 3, q_8, a), (q_7, 3, q_8, r)\}$,

and

$F = \{q_8\}$.

where, $Q$ is the set of states, $\Sigma$ and $\Delta$ are the sets of input and output symbols respectively,
\( \delta \) is the set of transitions in the form (start state, input symbol, next state, output symbol), \( q_0 \) is the initial state and \( F \) is the set of final states.

We can see that the corresponding left automaton (i.e. the automaton obtained by considering only the input symbols as transition labels) is nondeterministic. In the process of transformation from input to output strings, for the input symbol 1 the output is chosen randomly between the possible output symbols a and c, again a random choice (a or g) is made in answer to input symbol 2, and for the input symbol 3 the output depends on the output given to input symbol 2.

**Recurrent neural network**

The neural network, used in the experiment, has the basic Simple Recurrent Network (Elman network) structure, where the context layer keeps a history of the hidden layer’s activation states, and an additional context layer (named context-plus) as shown on Fig. 2.9.

The input and output layers have number of neurons equal to the number of input and output symbols respectively. We adopted localist representations of the input and output patterns. Thus, for example, the input 0 is represented in the input layer by having only the first neuron activated, respectively, the output m is represented by activating only the first neuron in the output layer.

The specific implementation is based on the LEABRA framework [41], which provides combination of error-driven and hebbian learning. The built-in inhibitory competition mechanism allows control over the size of the sparse distributed activation patterns in
the hidden layer. Furthermore, this competition mechanism is essential to obtain randomly one of the possible outputs. In our example, in response to input 1, the network should activate (randomly chosen) only one of the possible outputs (a or c). Without the competition mechanism, based only on the training examples, both of the outputs would be equally activated.

The hidden layer size was chosen, having in mind the number of distinct input-output pairs and leaving some room for varying the size of the activation patterns. At this stage, we were looking not for a minimal (in terms of number of neurons and connections) solution of the problem, but rather for a working prototype. Context and context-plus layers have the same size as the hidden layer.

The connectivity among the layers is as follows. Initially in the experiment, we used input layer fully connected to the hidden layer and later we switched to tessellated connectivity pattern. The hidden layer is fully connected to the output layer. Same is the connection from context to hidden layer. On the other hand, hidden to context layer and context-plus to hidden layer connections are such that, each neuron in the sending layer is connected to the corresponding neuron in the receiving layer. These one-to-one connections have fixed weights and do not participate in the learning process.

2.2.3 Method

After training with all sequences, we analyse the network’s behaviour and identify the key points affecting the output sequences, and then influence the network at these points.
Relying on the properties of the Leabra framework, we can expect the neural network to develop competing activation patterns in the hidden layer for the inputs having two possible outputs. For input 3, the context layer’s contents should provide enough information to resolve the competition, while for the inputs 1 and 2 the random noise in the activation function of the neurons will be the deciding factor.

By setting appropriate activation patterns in the additional context layer (context-plus), we will try to influence the outcome of the competition in the hidden layer, in the cases of inputs 1 and 2. In result, we hope to be able to select in advance which of the four sequences will be generated.

In the experiment, we proceed following the steps:

- Train the neural network from Fig. 2.9 with the four sequences of input-output pairs from table 2.3.

- After successful training, test the network by presenting the input sequence “010203” several times (until we obtain all distinct sequences) and save the hidden layer’s activation patterns.

- Analyse the saved patterns looking for similarity. It is done by clustering the patterns using Euclidean distance $d$:

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

where $n$ is the number of elements in the pattern, $x_i$ and $y_i$ are the corresponding elements in the two patterns being compared.

- If the resulting clusters are well formed and correspond to the input-output pairs, use combinations of the patterns for 1/a, 1/c, 2/a, and 2/g in the context-plus layer for testing. For example, to obtain the first sequence from table 2.3, we set the activation state of the context-plus layer to be the combination of 1/a and 2/a patterns. The context-plus layer’s activation state is held constant throughout the whole sequence. Although we need it only for the first and the third steps of a sequence, we want to test also the level of interference caused in the other steps.
• Repeat the successful tests from the previous step, using the neural network as a controller for the simulated robot.

2.2.4 Experiments

At first, in the experiment we used a network with the input layer fully connected to the hidden layer. After the training, we tested the network and saved the hidden layer’s activation patterns. The cluster plot, showing the distance among the patterns is in Fig. 2.10 (only one sequence from each type is included for better readability).

The cluster plot can be interpreted in the following way. Each pattern is named with a combination containing the number of the test sequence and the step number within the sequence, followed by the starting state (from the transducer in Fig. 2.8), the input symbol, the output symbol, and the next state. The different groups are shown as horizontal leaves off of a vertical branch that denotes the grouping. The Y axis is an index across the different patterns, while the X axis shows the distance among the patterns.

From the figure, we can see that the hidden layer’s activation patterns in this case are not well separated (compare the grouping for 3-a and 3-r with that from Fig. 2.11). This causes unwanted interference leading to the errors shown in table 2.4. Both cases with errors are the network being unable to produce the right output for the input 3. Since these outputs rely on the history kept in the context layer, they are much more sensitive to the (unnecessary in this case) influence from the context-plus layer. Even after attempts to manually adjust the relative influence from the input, context, and the context-plus layers through the weight scale parameters, we couldn’t neutralise that interference.

To achieve better separation between the patterns developed in the hidden layer, we used another variant of the neural network, with tessellated connection pattern from the input to the hidden layer. This time, each neuron in the input layer was connected to a separate column of neurons in the hidden layer.

Again, after training, the test run patterns from the hidden layer were saved. The produced cluster plot is in Fig. 2.11, while the patterns being clustered are in Fig. 2.12.

The results of 16 test sequences for each setting of the context-plus layer are in table 2.5. Without using the context-plus layer, the neural network chooses randomly between the
Figure 2.10: Cluster plot of the hidden layer’s activation patterns for the four sequences, obtained from the network with fully connected input to hidden layer.
Table 2.4: Results from 12 test sequences, for the case with fully connected input to hidden layer. The first column shows the contents of the context-plus layer, the next four columns show the how many times the corresponding input-output pair occurs, and the last column gives the number and type of the errors.

acceptable outputs as can be confirmed by the first row in the table. Setting the context-plus layer’s activation state to a combination of patterns identifying the desired sequence, results in that sequence being generated by the network in response to the sequence of inputs. For example, in the second row of the table, the activation state of context-plus layer is composed from the patterns for 1/a and 2/a. Then, when the input to the network is 1, the competition between the two patterns (e.g. second column, rows two and three in the Fig. 2.12) in the hidden layer is biased toward the pattern for 1/a, and the output settles to a. In a similar manner, the output a is generated for input 2. The interference with the patterns in the other steps seems small enough to allow generating the 16 sequences without errors.

Next, the same neural network was used to control the simulated robot from Fig. 2.7. Based on the results from 10 test runs for each sequence (shown in table 2.6), the network generated the desired robot behaviour most of the time. There were only two errors, when the robot crossed the line instead of avoiding it (row 2) and the robot attempted to release an object while the gripper was empty (row 3).

Table 2.5: Results from 16 test sequences, for the case with tessellated connection from the input to the hidden layer. See the explanation for table 2.4.
Figure 2.11: Cluster plot of the hidden layer’s activation patterns for the four sequences, obtained from the network with tessellated connection from the input to the hidden layer.
Table 2.6: Test results from using the neural network to control the simulated robot. The second column shows the number of successful runs out of 10, and the last column shows what error occurred.

<table>
<thead>
<tr>
<th>context-plus</th>
<th>good/all</th>
<th>error type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/a + 2/a</td>
<td>10/10</td>
<td>1/c instead of 1/a</td>
</tr>
<tr>
<td>1/a + 2/g</td>
<td>9/10</td>
<td>3/r instead of 3/a</td>
</tr>
<tr>
<td>1/c + 2/a</td>
<td>9/10</td>
<td></td>
</tr>
<tr>
<td>1/c + 2/g</td>
<td>10/10</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.12: The hidden layer’s activation patterns for the four sequences. Rows correspond to the sequences and columns correspond to the steps within a sequence. The activation of a neuron is proportional to the filled area.
2.2.5 Discussion

We investigated an approach to handle a specific case when the current inputs of a recurrent neural network together with the state retained in the context layer do not provide enough information to distinguish among the possible outputs. For instance, such problem arises when a neural network, used as a robot controller, have to deal with changing task requirements during execution and the task at hand cannot be identified directly from the inputs. Thus, the network should be able to switch easily among the possible outputs in order to adapt.

In the presented example, a neural network was trained to respond with alternative output sequences to a fixed input sequence. We were looking for a way to influence the generated output sequence without retraining the network (i.e., without changing the weights that encode the obtained input-output mappings). The investigated approach relies on an additional context layer (with fixed weights), to bias the random choices made by the network. To obtain the necessary activation states for the additional context layer, we analysed the hidden layer activation patterns and identified the ones that participate in the random choices. In the end, we were able to confirm that we can reliably select the desired output sequence by setting the additional context layer to appropriate activation state. Thus, the switching among the different output sequences is reduced to performing the desired sequence once, while copying the hidden layer activation states (at the specific choice steps needed to identify the sequence) to the additional context layer.

Here, although we used the simplest mechanism - setting the desired patterns in the context-plus layer and keeping them throughout the whole sequence, the results from the experiment were encouraging. There are two directions for reducing possible interferences among the activation patterns: to obtain well separated patterns and to add a control mechanism restricting the use of the context-plus layer only to steps in which it is needed.

At the current stage, the additional context layer’s activation states are set manually, but our intention is to adopt for that purpose the reinforcement learning mechanism described in [44].
2.3  Computational Model and Algorithm of Human Planning

2.3.1  Introduction

Using neural networks, it is relatively easy to learn separately simple mobile robot behaviours like approaching, wall following, etc., and with appropriate network architectures, combinations of such behaviours can be learned too. However, since these combinations are encoded into the network weights, switching from one combination to another often requires retraining. An interesting approach addressing the problem of switching among different mappings is presented in a working memory model proposed in [39]. It comes from the field of computational neuroscience and is a computational model of the working memory based on the prefrontal cortex (PFC) and basal ganglia. An important aspect of applying this model to learn a combination of behaviours is that the information for that combination is maintained explicitly as activation patterns in the PFC. Compared to a weights based encoding, these activation patterns can be updated faster and thus switching among possible combinations becomes easier.

In this section, an algorithm based on modification of that working memory model to include also an environment model is applied to a five-state random walk task. The environment model is added to provide model-based learning, motivated by the fact that reinforcement learning based only on real experience is associated with high costs (in terms of time, energy, etc.) when applied to real robots. Using additional model-generated or imagined experience helps to decrease the associated costs and also provides a link to planning, since, as argued in [49], planning can also be interpreted as learning from imagined experience. In light of this interpretation, the information (about the learned specific combination of behaviours) maintained in the working memory can be viewed as a simple plan to achieve the rewarded goal state. We would like to note that unlike the traditional symbolic planners, where the plan is perfectly remembered at the time of reaching the goal state, the implementation based on neural networks requires several passes through the desired sequence of actions. Because of this, we also consider that process as mental rehearsal.
2.3.2 Related Works

While simple mobile robot behaviours can be learned with feed-forward neural networks, combinations of behaviours, where sometimes identical sensory inputs should trigger different actions, require additional coordinating mechanisms. For example, in [7] a Khepera robot is trained to perform a garbage collecting task and the authors find a correspondence between specific behaviours and the evolved neural network modules. The interaction among these modules is controlled by selector neurons that give precedence of a given module over the others.

In contrast to the above work, where the modules are physically separate entities, Ziemke [59] interprets the trained Recurrent Neural Network (RNN) as a diachronically structured controller. In this case, instead of modules existing separately at the same time, a monolithic neural network instantiates different input-output mappings at various time points. An important aspect of the mechanism by which RNN achieve modularity is discussed in [14], where the switching between two input-output mappings is achieved by attentional control (attention is viewed as "an additional source of input that provides contextual support for the processing of signals within a selected pathway" (p. 335)). In RNN, the source that provides contextual support favouring one of the competing input-output mappings is the context layer. The state maintained in the context layer disambiguates the inputs and thus different outputs can be obtained for similar inputs.

Since, in RNN, the internal state plays a central role in switching between the alternative input-output mappings, the flexibility of updating and maintaining this internal state affects directly the flexibility of the resulting robot behaviours implemented by the network. The potential of the computational model of working memory based on the PFC and basal ganglia (PBWM model), proposed in [39], to provide such flexibility motivated us to investigate its application to learning combinations of robot behaviours.

2.3.3 Approach

In the presented approach, the PBWM model is used to implement several possible input output mappings and then to learn specific combinations. Also, a model of the environment is added to provide imagined experience. We are interested in two consequences of
using an environment model: lowering the costs associated with actually performing the actions and extending the neural network model to a planning system supporting grounded representations.

**Working Memory Model**

Here we present an outline of the PBWM model (refer to [39] for details). The model implementation is based on the Leabra framework [41], uses point neuron activation function for modelling the neurons, k-Winners-Take-All inhibition to model competition among the neurons in a layer, and a combination of Hebbian and error-driven learning.

The neural network structure (Fig. 2.13 c) consists of two groups of layers. The first group includes the Input, Hidden, Output, NextInput, and PFC layers. The NextInput layer is used for the environment model and will be explained later. The Input, Hidden, and Output layers form a standard three-layer neural network structure. The PFC layer is an improved context layer, which is bi-directionally connected with the Hidden layer, and influences the input-output pathways. The PFC layer is divided into stripes to allow independent control over the updating and maintenance of parts of the activation state. The rest of the layers form the second group, which implements a gating mechanism for control over the updating and maintenance of the PFC activation state. Generally, a positive reward leads to stabilising of the current PFC activation state, while a negative reward results in updating (a part of it) and establishing of another state.

**Model of the Environment**

Under the reinforcement learning framework (see Fig. 2.13 a)), an Agent performs an action \( a \) based on the current sensory input and the policy formed so far. The Environment (or the Environment Model) responds with a new sensory input \( s \) and an external reward \( r \). The Agent adjusts its policy based on the reward and completes the cycle by performing a new action.

The two parts of the environment model are implemented as follows. The model of the next input is implemented as an additional output layer, trained to predict the next input based on information from the current network state. The model of the external reward at
this stage is implemented outside of the network as a simple lookup table keeping the last reward received for each input-output pair.

2.3.4 Simulation

A five-state random walk task was used to test the approach. In this task, there are five squares in a row, and an agent that moves one square left or right. The start position is the middle square and a move outside from the leftmost and rightmost squares sends the agent back to the start position. Two goals were used: moving right from the rightmost square and moving left from the leftmost square. Figure 2.13 b) shows the settings and a finite state automaton describing the states and the transitions (inputs $i$ and outputs $o$ in the network). The reward value corresponds to goal set to the right side.

For this simulation, we used the PDP++ neural network simulator\(^1\). The network input (see Fig. 2.13 c)) is the current position of the agent. The network outputs are the current action in the Output layer and the prediction of the next input in the NextInput layer. The Hidden layer has one neuron for each state-action combination. The top row encodes move-right and the bottom row encodes move-left. A restriction is imposed through the k-Winners-Take-All function to allow only one active neuron. The weights between the Input, Hidden, Output, and NextInput layers are hand-coded (in a separate experiment we have confirmed that these weights can be learned too) so that from each state the two possible actions are equally probable. The PFC has 8 stripes, each one with the same size as the Hidden layer. The Hidden layer has one-to-one connections with each stripe in the PFC layer.

The training process, inspired by the Dyna algorithm [49], is an interleaving execution of two loops. One for the real experience, receiving the next input and the external reward from the environment and the other, for the model-generated experience, obtaining the input from the NextInput layer and the external reward from the lookup table. Two groups of simulations were performed: with and without model-generated experience. In each group, there were two simulations: with the goal on the right and on the left. After training for 300 sequences of real experience, a test consisting of 10 trials, 50 sequences each, was

\(^1\)PDP++ software package, ver. 3.2a, http://www.cnbc.cmu.edu/PDP++/PDP++.html
Figure 2.13: a) Reinforcement learning with additional model-generated experience. b) Random walk task settings. c) Neural network structure.
performed. The test results are summarised in Fig. 2.14.

An additional test with switching between the two goals were performed. The plot in Fig. 2.15 shows the results of consecutive training with switching the goal (between left and right) every 100 sequences. Both, real and model-generated experience, were used in the training process. The horizontal axis shows the sequence number and the vertical axis shows the length of the corresponding sequence in steps. A maximum of 20 steps were allowed for a sequence. If the goal state was not reached within 20 steps, the sequence was stopped and the next sequence was started from the initial state. This restriction was used to avoid endless cycling between few states. As the plot shows, less than 50 sequences were needed to switch from one goal to the other. It is important to note that the weights between the Input, Hidden, and Output layers are fixed, and the information for the current goal and how to achieve it is kept as sustained patterns of activity in the PFC layer. For example, in Fig. 2.13, c) the active neurons in the PFC layer encode move-right from the input states \(i0, i2, i3, \) and \(i4,\) i.e., how to achieve the goal on the right.

### 2.3.5 Discussion

From the simulation results in Fig. 2.14, it can be seen that the neural network learned to achieve the goal state. Also, the neural network trained with additional model-generated experience performs better than the one trained only with real experience. The desired functionality for fast switching among learned behaviours was confirmed in the simulation test with switching the goal between left and right. These results were obtained using only the reward as a teaching signal. On the other hand, if we use supervised learning, i.e., give not only the reward signal but also the desired network output (target output), the switching between the alternative behaviours occurs in less than five sequences (confirmed in separate simulations). The problem is that supervised learning is not suitable for planning, since in general, we do not know the target outputs at each step. A middle ground solution between pure reinforcement learning and supervised learning could be implemented to improve the network performance. The idea is to use a randomly created target output, which is different from the network output, when a negative reward is received. In this way, the network would be forced to change the current output and explore different one.
Figure 2.14: Plot of the average number and standard deviation of left and right sequences over the 10 test trials. The horizontal axis shows the settings for the four simulations in the form: with and without mental rehearsal, each one with the goal on the right and on the left.
Figure 2.15: Plot of the number of steps used to achieve the specified goal for each training sequence. In this training process, model-generated experience was used, and the goal was shifted every 100 sequences.
Another important result is evident from the obtained activation patterns in the PFC layer. The neural network shown in Fig. 2.13 c), has been trained to achieve the goal state on the right side. As can be seen, mostly active are the units in the top row of the PFC stripes. They correspond to the units for move-right in the Hidden layer and consequently, bias the neural network output to prefer this action in each state. Thus, the contents of the PFC layer can be interpreted as a simple plan (a combination of actions) leading to the goal state. The future work includes using distributed representations in the network, learning the weights between the input, hidden and output layers, and using the model to control a real robot.

2.4 Summary

In this chapter, we proposed an algorithm providing mental rehearsal functionality that supports fast switching among previously learned behaviours. The algorithm is based on two fundamental ideas. The first idea is to encode the robot’s behaviour in the context layer of a recurrent neural network by adding memory of future sensory states in addition to the past ones as illustrated in Fig. 2.16. The benefit is that the behaviour encoded in the context layer can be changed faster than behaviour encoded only in the neural network weights (as in normal feed forward networks). But since the behaviour changes still require a number of repetitions, the second idea is to implement ‘mental rehearsal’ or learning from imagined experience. This is illustrated in Fig. 2.17 with two interleaving loops of real and imagined experience. Thus, the number of required repetitions actually performed in the real world is decreased further. The smaller number of real world repetitions results in faster behaviour switching.
Figure 2.16: Using working memory model for encoding the past and future states.

Figure 2.17: Adding an imagined experience loop for implementing mental rehearsal.
Chapter 3

Learning from Masterpieces

We can learn from good examples or successful solutions provided by skilled individuals. An example situation is shown in Fig. 3.1, where a person copies a masterpiece from a famous painter. When doing this, the goal is not the final copy but to acquire, in the process of recreating the initial painting, some of the knowledge, the understanding, the skills that allowed the master to create the painting in first place.

Figure 3.1: Example of learning from masterpieces.

In this chapter we propose an algorithm for learning task solving strategies from positive examples only. Similarly to the previous chapter, we use the idea to encode the desired
strategy in the working memory. The new idea here, is in adding a modification to allow learning from positive example only.

A simplified illustration of the idea is given in Fig. 3.2. Pane a) shows a desired solution that can be obtained if we use both positive and negative training examples. Pane b) shows a solution that can be obtained by starting with random initial model parameters and learning from positive examples only. The initial model parameters (in this case, weights) determine the initial position and orientation of the decision boundary. It is thus possible to start from such position that all positive examples are already on the positive side of the decision boundary. In this case, since there will not be any errors, the model weights will not change and the decision boundary will remain in the initial position. But if we compare with the desired solution from pane a), we can see that there are negative examples that fall on the positive side of the decision boundary, consequently that solution is not desirable. To avoid this problem, we implement the following idea. Instead of starting with random model parameters, we set them so that initially all possible examples fall on the negative side of the decision boundary, as shown at the top of pane c). Then, with each new positive example, the learning process adjusts the weights and the decision boundary shifts accordingly to a new position, leaving that example on the positive side. In this way, while the final solution may still classify some of the negative examples incorrectly, the number of such mistakes would be less that the solutions obtained with random initial parameters.

3.1 Cognitive Control Based Realisation of Human Skill Algorithms

3.1.1 Introduction

Often, real-life problems are treated as optimisation problems to obtain good solutions, applying to practice well studied theoretical approaches [16]. But in some cases, the problems are very difficult to formulate mathematically, in the sense that the objective function and the relevant constraints are vague and cannot be described completely in a strict form. The necessary knowledge about the relevant constraints is accumulated in the process of repeatedly generating and evaluating solutions of concrete problem instances in various
Figure 3.2: Illustration of learning from positive examples only. a) desired solution, b) possible non-optimal solution that can be obtained if starting with random model parameters, c) using an initial bias set to reject all positive examples and gradually expanding the area if the new examples require it.
situations. Skilled practitioners have such experience and possess good understanding of the constraints, which allows them to provide sufficiently good solutions, tailored to the specifics of the problem instances at hand. Unfortunately, part of their knowledge is “passive”, i.e., they are not able to enumerate all constraints at will, but can use them successfully when faced with a concrete problem that provides appropriate cues. Nevertheless, we can identify skilled humans that consistently provide good solutions, even in the case of problems that are difficult to formalise.

The purpose of this research is to find good algorithms for solving real-life optimisation problems for which the objective function and the constraints are difficult to formulate. Besides the traditional approaches, several neural network models have been used to obtain optimal or quasi-optimal solutions [37, 48, 56]. Unlike them, we follow a different approach trying to realise algorithms applied by skilled individuals in solving such problems instead of attempting to formulate and solve the corresponding mathematical optimisation problems. With such approach, we are concerned less about the strict optimality of the obtained solutions and more with acquiring a rich repertoire of domain specific, useful strategies and the algorithms or mechanisms that support them.

One problem that arises when using the above approach is that it is difficult to obtain negative examples for some problems. In such cases, suitable algorithms that can learn from positive examples only are necessary. Here, we propose an algorithm for learning strategies from human solved problem instances, using positive examples only. The proposed algorithm is based on a modified neural network implementation of a cognitive control model. We present results from an experiment comparing the performance with two other algorithms, used often for learning from positive examples.

After description of the problem in the next section, we introduce the initial form of the used cognitive control model and the proposed modifications. Then, follows a description of the algorithms used in the comparison and the performed experiment. The results of the experiment are discussed in the last section.
3.1.2 Problem Description

In this section we consider the problem of learning from positive examples only. This problem can be formulated as follows. Given a set of input-output pairs \((x_i, y_i), i = 1, ..., l\), where \(l\) is the number of pairs, \(x_i \in \mathbb{R}^n\) is a vector of real-valued input attributes, and \(y_i \in \{0, 1\}\) is a class label, we want to find a function of the form \(c : X \rightarrow \{0, 1\}\) so that \(c(x_i) = y_i\) for all \(i\).

The pairs for which \(y_i = 1\) are called positive examples and the pairs for which \(y_i = 0\) are called negative examples. Since we consider the case of learning from positive examples only, the training data set consists of examples in the form \((x_i, y_i)\), where \(y_i = 1\).

Problems of this type are common when positive examples are readily available, while negative examples are difficult or impossible to obtain explicitly. For example, if we want to learn the class of web pages that are interesting to a given subject, we can obtain easily positive examples from the subject’s bookmarks, while to obtain negative examples, i.e. uninteresting web pages, we have to ask him to classify pages explicitly.

Thus, using only examples of the type \((x_i, 1)\), the learning algorithm should find a function \(c(x)\) that would handle correctly the \((x_i, 0)\) cases too. To achieve this, the learning algorithm has to avoid the common problem of over-generalisation. Over-generalisation happens when the training error is minimised by simply classifying all possible inputs as members of class 1.

In this work, we use a packing problem as an example. From a set of solved problem instances, we want to learn the strategy that was used to obtain the solutions. Following the above problem formulation, the strategy is expressed as the function \(c(x)\) that assigns the items to one of two classes: items to be put into the container \((y_i = 1)\) and items to be left out \((y_i = 0)\). The input vector \(x_i\) is a vector of item features.

We treat the learning problem as learning from positive examples only, since collecting such data requires only access to solved problem instances and thus simplifies the process of data acquisition in the real-world case. Positive examples are the items put into the container and can be easily retrieved from the problem solutions. On the other hand, to obtain negative examples, i.e., the items left out of the container, we need additional information besides the packing problem solutions.
To summarise, the problem we want to solve is to learn human strategies from solved packing problem instances, using positive examples only. In this initial stage, the proposed algorithm is used to select the items to be put into the container based on their properties. The positions and the orientations of the items in the container will be considered at a later stage.

3.1.3 Proposed Algorithm

In the proposed algorithm, we use a particular neural network implementation of a cognitive control model, namely the Prefrontal cortex and Basal ganglia Working Memory model (PBWM) [40]. A brief description of this model is given below, followed by the explanation of the modifications made to use the model in the considered task.

Outline of the PBWM Model

The prefrontal cortex (PFC) is believed to play an important role in the formation of complex, goal directed behaviours [15, 13]. The PBWM model is an attempt to model this functionality at computational level.

The model implementation is based on the Leabra framework [41] featuring point neuron activation function for modelling the neurons, k-Winners-Take-All (kWTA) inhibition to model competition among the neurons in a given layer, and a combination of Hebbian and error-driven learning.

A simplified structure of the model can be seen in Fig. 3.3. Besides the standard input, hidden, and output layers there is a PFC layer and a group of layers shown as Gating. The PFC layer serves as an improved context layer. It receives activation from and sends activation back to the hidden layer, influencing the input-output pathways. Unlike the context layer of a Simple Recurrent Neural network [17] which is updated at each step, the PFC layer updating is controlled by a specialised gating mechanism based on a reward signal and implemented in the group of layers named Gating.

For detailed explanation of the PBWM model, refer to O’Reilly and Frank [40].
Modification for Learning from Positive Examples

The structure of the modified neural network implementation is shown in Fig. 3.4. The small rectangles represent units. The units are grouped in layers with the layers’ names written beside. The three rectangles with a dropped shadow show groups of layers. The arrows show connections between the layers. The relevant layers, connections and the behaviour of the neural network are explained below.

Layers  The input of the network encodes an item, candidate to be put into the container. After deciding on the set of features characterising the item, we encode each feature in a separate input layer using 1 if a feature is present and 0 otherwise. In the presented experiment, we use the following seven item features: (item-id, max-value, min-value, max-weight, min-weight, max-size, and min-size). The input layers are named after the features they encode. They are shown in the group named Features in Fig. 3.4, except the item-id layer, which is the first layer on the left. The input layers encoding the position and the orientation of the current item are shown in the group Not used. They are intended for further development and were not used in this experiment.

The output layer of the network encodes whether the item presented to the input should be put into the container or not. Therefore, the output layer has two units, one for put and
the other for reject (see Fig. 3.5).

The hidden layer, has one unit for encoding the reject action, and the rest of the units encode the input item features. Figure 3.5 shows what is encoded by the hidden layer units.

The PFC layer has six stripes (corresponding to the rows of units in the PFC layer shown in Fig. 3.4) each one having the same structure as the hidden layer.

The rest of the layers included in the group Gating, their connections and behaviour are the default ones from the PBWM model.

Figure 3.4: Structure of the PBWM neural network model, modified for the packing task.

**Connections** The current implementation uses preset weights for the connections among the input, hidden, and output layers. Each of the input layers is connected to the units in the hidden layer encoding the corresponding item feature (see Figs. 3.5). An exception is the input layer encoding the item id. This input layer is connected to all units in the hidden layer.
The connections between the hidden and output layers are bidirectional, i.e., there are symmetric connections from the hidden layer to the output layer and from the output layer to the hidden layer. The unit encoding the reject action in the output layer is connected to the unit encoding the reject action in the hidden layer. The unit encoding the put action in the output layer is connected to the rest of the units in the hidden layer.

The connections among the hidden and the PFC layers are also bidirectional. Each unit in the hidden layer is connected with the units in the corresponding position of the six PFC stripes.

**Initial Bias Implementation**  An initial bias is used to implement the idea described in the beginning of this chapter, namely, to ensure that at the beginning of the learning process, the neural network classifies all training examples as negative.

The initial values of the preset weights and the corresponding learning rates are given in Table 3.1. A drawing of the preset weights relative to the hidden layer is shown in Fig. 3.6. The rest of the parameters are the default ones from the PBWM model.

The initial bias toward rejecting the items presented to the network input is implemented by presetting the weights \( w_{id1} \) from the item-id input layer to the hidden layer unit encoding reject action to 0.72 and the weights \( w_{id2} \) to rest of the hidden layer units to 0.38. The weights \( w_f \) from the input layers max-value, min-value, max-weight, min-weight, max-size, and min-size to the corresponding hidden layer units are set to 0.30. The weights \( w_{pfc2} \)
Figure 3.6: Preset weights. Only the first two units of the hidden layer are shown. The rest of the units are identical to the max-value unit. Details of the weights are given in Table 3.1.

The interpretation of these inequalities is as follows. The input layers max-value, min-value, max-weight, min-weight, max-size, and min-size encode the presence of the corresponding feature for the current item. The PFC layer units encode the prevalent features of the items in the class put (the explanation for this is given in the next subsection). The item-id layer is used as a source for the bias. In the hidden layer, there is a competition among the units, implemented by the winner-takes-all mechanism of the network. To illustrate the competition, we show the first two units in the hidden layer in Fig. 3.6. The first inequality Eq. 3.1 shows the case when feature max-value is not present in the current item, neither it is among the prevalent features encoded in the PFC layer. In this case, the

\[
\begin{align*}
    w_{id1} & > w_{id2} \\
    w_{id1} & > w_{id2} + w_f \\
    w_{id1} & > w_{id2} + w_{pfc2} \\
    w_{id1} & < w_{id2} + w_f + w_{pfc2}
\end{align*}
\]
Table 3.1: Initial values of the preset weights and the corresponding learning rates.

<table>
<thead>
<tr>
<th>weight</th>
<th>From</th>
<th>To</th>
<th>Initial value</th>
<th>Learn. rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{id1}$</td>
<td>item-id layer</td>
<td>hidden layer, reject unit</td>
<td>0.72</td>
<td>0.0</td>
</tr>
<tr>
<td>$w_{id2}$</td>
<td>item-id layer</td>
<td>hidden layer, the other units</td>
<td>0.38</td>
<td>0.0</td>
</tr>
<tr>
<td>$w_f$</td>
<td>max-value layer</td>
<td>hidden layer, max-value unit</td>
<td>0.30</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>min-value layer</td>
<td>hidden layer, min-value unit</td>
<td>0.30</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>max-weight layer</td>
<td>hidden layer, max-weight unit</td>
<td>0.30</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>min-weight layer</td>
<td>hidden layer, min-weight unit</td>
<td>0.30</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>max-size layer</td>
<td>hidden layer, max-size unit</td>
<td>0.30</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>min-size layer</td>
<td>hidden layer, min-size unit</td>
<td>0.30</td>
<td>0.0</td>
</tr>
<tr>
<td>$w_{pfc1}$</td>
<td>hidden layer</td>
<td>PFC layer</td>
<td>0.25 ± 0.05</td>
<td>0.0</td>
</tr>
<tr>
<td>$w_{pfc2}$</td>
<td>PFC layer</td>
<td>hidden layer</td>
<td>0.25</td>
<td>0.001</td>
</tr>
<tr>
<td>$w_o$</td>
<td>hidden layer, reject unit</td>
<td>output layer, reject unit</td>
<td>0.25</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>hidden layer, the other units</td>
<td>output layer, put unit</td>
<td>0.25</td>
<td>0.0</td>
</tr>
</tbody>
</table>

The initial weights were obtained through a simplified grid search using $w_{id1}$ and $w_{id2}$ around a starting point of $w_{id1} = 0.70$ and $w_{id2} = 0.35$, with step 0.01.

**Neural Network Behaviour**  The parameter controlling the maximum number of active units in the hidden layer is set to allow only one unit, i.e, implementing a winner-takes-all competition.

When an item is presented to the network input and there are no active units maintained
in the PFC layer, due to the bias described above the hidden layer unit encoding reject action becomes activated and in turn activates the reject action in the output layer. Thus, the item is rejected.

In the case when an item feature, for example max-value, is presented in the input and simultaneously some of the units encoding the same feature in the PFC layer are maintained active, the hidden layer unit encoding the max-value feature receives activation both from the input and from the PFC. In result, this hidden layer unit overcomes the bias toward the reject unit, wins the competition and becomes activated. Consequently, the output layer unit encoding the put action becomes activated and the item is classified as suitable to be put into the container.

Which units in the PFC layer are maintained active? During the training process, some of the randomly formed activation patterns (‘activation pattern’ refers to the pattern formed by the activation levels of all neurons in a given layer at a certain time step) in the PFC layer happen to support a neuron in the hidden layer that encodes certain feature present in the current item. Then, this neuron wins the competition in the hidden layer, becomes activated, and the network output is set to put. The obtained positive reward reinforces the maintenance of that PFC pattern through the gating mechanism. On the other hand, if a PFC pattern does not support any of the present features of the current item, the network output is reject and the lack of reward (since we use only positive examples in the training process) leads to destabilisation of that pattern and establishing of another one.

In result of this process, the patterns maintained in the PFC layer become associated with the predominant features (or their combination) present in the items selected for placement in the container. During the testing, when presented with new items, the items without these features will be rejected while the items having them will be placed in the container.

In the reported tests, we used the PDP++ neural network simulator \(^1\).

\[3.1.4 \text{ Comparison Experiment} \]

In this experiment, the proposed algorithm, based on a modified PBWM model, is compared to one-class Supporting Vector Machines (SVM) and auto-encoding neural network

\(^1\text{PDP++ software package, version 3.2a, http://www.cnbc.cmu.edu/PDP++/PDP++.html} \)

52
algorithms, which are used often in learning from positive examples only. These algorithms are outlined below. The data for training and testing the models were obtained from human solved instances of a two-dimensional packing problem. The purpose of the comparison is to confirm the viability of the proposed approach and to identify aspects that need improvement.

Algorithms Used for Comparison

One-Class SVM  Here, we outline briefly the one-class SVM, introduced by Schölkopf [46]. This algorithm has been widely used to learn from positive examples in document classification [30], computer security [53], and robot vision [43].

When trained with a set of data points, the one-class SVM algorithm learns a function that has value $+1$ in the region containing most of the data points, and value $-1$ outside of this region. The learned function represents a hyperplane in the feature space. The problem is to find such hyperplane in the feature space that separates the data points from the origin.

Let us consider the training data $x_1, \ldots, x_\ell \in X$, where $\ell$ is the number of observations, and a feature map $\Phi : X \rightarrow F$, which maps points from the input space $X$ to points in the feature space $F$. The maximum margin solution of the one-class SVM problem is obtained by solving the following optimisation problem:

\[
\begin{align*}
\min_{w \in F, \xi \in R^\ell, \rho \in R} & \quad \frac{1}{2} \| w \|^2 + \frac{1}{\sqrt{\ell}} \sum_{i} \xi_i - \rho \\
\text{subject to} & \quad (w \cdot \Phi(x_i)) \geq \rho - \xi_i, \xi_i \geq 0,
\end{align*}
\]

where $w \in F$ is a vector describing the hyperplane in the feature space, $\rho \in R$ is the margin of the hyperplane with respect to the data, $\xi_i$ are non-zero slack variables allowing for a soft margin, and $\nu \in (0, 1]$ is a parameter that represents an upper bound of the fraction of outliers in the data. The learned decision function is:

\[
f(x) = \text{sgn}((w \cdot \Phi(x)) - \rho),
\]

where $\text{sgn}(z)$ equals 1 for $z \geq 0$ and $-1$ otherwise.
Figure 3.7: Structure of auto-encoding neural network. The input and output layers have equal number of units \( d \), while the hidden layer has \( m \) units, where \( m < d \).

In the experiment, we used the LibSVM implementation [10] of the one-class SVM (through the e1071 library in the R statistics environment [26], http://www.r-project.org/) with the default Radial Basis Function (RBF) kernel.

**Auto-encoding Neural Network**  Auto-encoding neural networks have been applied for novelty detection, for example, in Japkowicz et al. [28]. Basically, they are three layer feed-forward neural networks, as shown in Fig. 3.7. Suppose that the input pattern is a vector of size \( d \), then the network has \( d \) units in the input and output layers and \( m \) units in the hidden layer, where \( m < d \).

The network is trained using backpropagation to learn the identity function by applying the same patterns as inputs and targets. Since the number of units in the hidden layer is smaller than that in the input and output layers, perfect reconstruction of all input patterns is impossible and the network learns to represent the input patterns in a lower dimensional sub-space retaining the essential characteristics. Using the trained network, if the result of presenting a new input pattern is obtaining the same pattern in the output, then we can conclude that the new pattern is similar to the training patterns and belongs to the same class. Conversely, if the output is different, then the new pattern does not belong to the class of patterns used for training.

In the experiment, we used the implementation provided by nnet library in the R statistics environment [26].
Data

For this experiment we used data from human solved instances of two-dimensional packing problem. Four subjects participated and each subject provided solutions to eight instances of the task.

The task is described in Fig. 3.8. A set of 10 items, randomly selected among the shown initial set of 15 items is presented to the subject. Each item has the following parameters: value, weight, and size. The subject selects the items to be placed into a 5 by 5 container according to his own strategy decided in advance. The following information is collected:

- which items are placed in the container and their parameters;
- the order in which the items are placed in the container;
- which items are left out of the container and their parameters.

The information for the items left out of the containers is collected for testing purposes in the present experiment. In the real-life situations, the testing should be based on the suitability of the generated solutions.

Based on the above information, the following item features are computed:

- \( \text{max-value} \) if the item has maximum value among the items waiting to be placed in the container;
- \( \text{max-value} \) if the item has minimum value among the items waiting to be placed in the container;
- \( \text{max-weight} \) if the item has maximum weight among the items waiting to be placed in the container;
- \( \text{max-weight} \) if the item has minimum weight among the items waiting to be placed in the container;
- \( \text{max-size} \) if the item has maximum size among the items waiting to be placed in the container;
• *max-size* if the item has minimum size among the items waiting to be placed in the container;

Each example in the data consists of an item’s id, item’s features and the item’s class. There are two item classes: *put* — the class of items put into the container; and *reject* — the class of the items left out of the container.

The items placed into the container are positive examples and are used both for training and testing. The items left out of the container are negative examples and are used only for testing, to evaluate the performance of the compared algorithms.

**Training and Testing Procedures**

Since the size of the available data is restricted, we used a variant of the hold out method [3, p.372] for obtaining an estimation of the generalisation achieved by the compared algorithms.

Considering the data from each solved problem instance as one set, each subject provided eight sets. Ten groups of training and testing data were generated by randomly selecting five of the sets for training and the rest three sets for testing. The negative examples, i.e., the items not put into the container were removed from the training data. Thus, the data sets for training contained items from the *put* class only, while the data sets for testing included items from both classes.

For each subject, we did the following.

We performed a grid search to find good values for the $\nu$ and $\gamma$ parameters of the one-class SVM algorithm, where $\nu$ is a parameter controlling the number of outliers included in the positive class and $\gamma$ is an RBF kernel parameter. The two parameters were varied in $(0, 1)$ interval with step 0.1. Then, with the obtained parameters we trained and tested the algorithm with each of the ten data groups described above. The results of the ten tests were used to evaluate the algorithm.

Similarly, we trained and tested the auto-encoding neural network with three, four, and five units in the hidden layer (the input and output layers have six units) to find a good value. A threshold of 0.5 was used to decide the class of the item presented to the network. If the output error was below that threshold, the item was assigned to the *put* class, otherwise to
Figure 3.8: A two-dimensional packing problem is solved by the subjects to obtain training and testing data. Below is shown an example solution. The name, value, and weight are written inside the items.
Table 3.2: Groups of items based on the real and assigned classes.

<table>
<thead>
<tr>
<th>Item groups</th>
<th>Real class</th>
<th>Assigned class</th>
</tr>
</thead>
<tbody>
<tr>
<td>true positives</td>
<td>put</td>
<td>put</td>
</tr>
<tr>
<td>false positives</td>
<td>reject</td>
<td>put</td>
</tr>
<tr>
<td>true negatives</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>false negatives</td>
<td>put</td>
<td>reject</td>
</tr>
</tbody>
</table>

the reject class. Then, we trained and tested the network with each of the ten data groups. Again, the results of the ten tests were used to evaluate the algorithm.

Using the parameters described earlier, the proposed algorithm was trained and tested with each of the ten data groups. The performance of the algorithm was evaluated using the results of the ten tests.

Training and testing with the one-class SVM algorithm and the auto-encoding neural network on one data group took on average less than half a minute, while the proposed algorithm required about five minutes. Overall, since we didn’t use algorithm implementations optimised for speed, we were not concerned with comparing the time requirements in the presented experiment.

Evaluation Criteria

To compare the performance of the tested algorithms, we used the $F_1$-measure based on the recall and precision values, as defined below.

Using the real class of an item and the class assigned by the tested algorithm, we define the four groups of items given in Table 3.2.

Recall $R_{put}$ and $R_{reject}$ measures are computed as:

\[
R_{put} = \frac{\text{Number of true positives}}{\text{Number of items from class put in the test set}}
\]

\[
R_{reject} = \frac{\text{Number of true negatives}}{\text{Number of items from class reject in the test set}}
\]
Table 3.3: Results of the comparison experiment.

<table>
<thead>
<tr>
<th>Subj.</th>
<th>One-class SVM</th>
<th></th>
<th>Auto-encoding NN</th>
<th></th>
<th>Proposed algorithm</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( F_1 ) reject</td>
<td>( F_1 ) put</td>
<td>( F_1 ) reject</td>
<td>( F_1 ) put</td>
<td>( F_1 ) reject</td>
<td>( F_1 ) put</td>
</tr>
<tr>
<td>1</td>
<td>0.51</td>
<td>0.72</td>
<td>0.90</td>
<td>0.96</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>0.48</td>
<td>0.69</td>
<td>0.94</td>
<td>0.97</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>0.52</td>
<td>0.43</td>
<td>0.33</td>
<td>0.73</td>
<td>0.73</td>
<td>0.87</td>
</tr>
<tr>
<td>4</td>
<td>0.68</td>
<td>0.69</td>
<td>0.86</td>
<td>0.92</td>
<td>0.98</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Precision \( P_{\text{put}} \) and \( P_{\text{reject}} \) measures are computed as:

\[
P_{\text{put}} = \frac{\text{Number of true positives}}{\text{Number of items assigned to class put}}
\]  
(3.10)

\[
P_{\text{reject}} = \frac{\text{Number of true negatives}}{\text{Number of items assigned to class reject}}
\]  
(3.11)

The recall measure gives an idea, for example, what fraction of the items that have to be put in the container have been put. The precision measure gives an idea of what fraction of the items put into the container have to be there. Taken separately, these two measures are not sufficient, since the recall measure for a given class may be maximised by simply assigning all items to that class, or high precision for a given class may be achieved by rarely assigning this class. For this reason, we use the \( F_1 \)-measure which combines both recall and precision.

\( F_1 \)-measure is computed as:

\[
F_1 = \frac{2RP}{R + P},
\]  
(3.12)

using the recall \( R \) and precision \( P \) values for the corresponding class.

The recall, precision, and \( F_1 \)-measure are bounded in the \([0, 1]\) interval and values closer to 1 indicate better performance.

**Results**

For each subject, the results from the ten tests for each algorithm were used to compute the \( F_1 \)-measures. Table 3.3 shows the mean values of the \( F_1 \)-measure computed over the ten
tests for each combination of algorithm, class, and subject.

It can be seen that the proposed algorithm performs better than the two other algorithms in all four cases, corresponding to the data from the four subjects.

Common problem with learning from positive examples only is the bad performance on the class not included in the training data. For the proposed algorithm, the performance on the reject class is only slightly worse than the performance on the put class.

If we compare the performance of the proposed algorithm shown on the separate subjects’ data, the performance on the data obtained from the third subject is worse than the others. The fact that the other two algorithms also show drop in the performance for the same dataset suggests that this is not due to characteristics of the proposed algorithm. One possible reason for the relatively poor performance may be that the selected item features are not well suited to express the strategy used by the third subject.

Discussion

Using a packing problem as an example, we performed an experiment in which the performance of the proposed algorithm is compared to that of one-class SVM and auto-encoding neural network algorithms. These two algorithms were selected because they are often used in learning problems where only positive examples are available.

The results of the experiment confirm that the proposed algorithm performs well and compares favourably to the one-class SVM and auto-encoding neural network algorithms for the type of strategies presented in the training data.

A possible reason for the relatively poor performance of the one-class SVM may be its sensitivity to the model parameters [10]. Although, in the experiment, we used the default parameters and performed a grid search over the two recommended parameters $\nu$ and $\gamma$, we suppose that better results can be achieved by additional fine tuning of the model.

The reported results support the viability of the proposed algorithm and further tests with more features and larger datasets are considered. In addition to scaling up the problem instances, we need to conduct tests with data obtained from skilled individuals. Although the subjects that took part in the presented experiment did not have specific experience or extensive knowledge in the packing problem domain, the obtained results are encouraging and we expect that the proposed algorithm will perform well on data from skilled
Further improvements of the proposed model concern including the position and orientation of the items as additional input features, as well as learning suitable combinations of the input features in the hidden and PFC layers to allow increasing complexity of the learned strategies.

## 3.1.5 Simulation: Generating Problem Solutions

In the previous experiment we compared the performance of the proposed algorithm with one-class SVM and auto-encoding neural networks. In this simulation, we use neural networks trained by the proposed algorithm to generate packing task solutions and compare them with the solutions that provided the training data.

### Adding a Selection Process Based on Performance Index

To select among the trained neural networks a good one for generating solutions, a selection process is used in addition to the training process, as outlined in Fig. 3.9.

We start with the assumption that an expert has some strategy (depicted as Rules) for solving the packing problem. Based on this strategy, when presented with instances of the packing problem he generates solutions (encoded as Input Output pairs). An estimation of the quality of these solutions is given by some Performance index and is used by the expert to improve his strategy.

A set of solved packing problem instances (Training data) are used to train the neural network model, saving the network state at regular intervals during the training process when the training error falls below some threshold value. The resulting models are depicted as NN models.

Since there are only positive examples in the training data set, we cannot split it and use one part for testing. That is because the testing still won’t reveal the problem arising when the networks accept all presented items. Thus, we have to test the networks by solving instances of the initial problem. The solutions produced in this way are labelled Testing data.

In the general case, we do not know the target outputs (i.e., to accept or to reject the
Figure 3.9: Outline of the training and selection processes.
item presented to the input) in the testing process, so we cannot compute the testing error directly. Therefore, we need some other measure to evaluate the performance of the tested networks. Such measure is given by the Performance index computed over these solutions. It is used in the selection process to choose the best performing model, which becomes the Final NN model.

In the simulation presented below, we choose a performance index and rules that produce good solutions. Based on these rules, we generate the training data. Then, we train the neural network and test the saved instances. Since in this case we know the initial rules, we can compute the testing errors to check later against the results obtained through the performance index. We compute the performance index for the testing data for each network and use the results to select the best one among the saved networks. If the performance index is suitable for selecting a good network, then the chosen network should be the same as the one selected through the testing errors.

Conditions

The problem used in the simulation was a two-dimensional packing problem described as follows. A container with size of 5 by 5 units and an initial set of 15 items, with size ranging from 1 by 1 units to 3 by 3 units were used (see Fig. 3.8). The value and the weight of the items were in the intervals between 0.0 and 1.0 and between 0 and 10 respectively.

The task used to generate the training data was to fit as much summary value in the container as possible. In other words, the performance index was given by the sum of the value of the items packed into the container and optionally the sum of their size.

A basic strategy placing the most valuable item first at the first fitting position and orientation starting from the bottom left corner was implemented in a program and used to generate 100 solutions of the packing problem which were used as training data.

Neural Network Parameters

In this simulation we used the same structure and parameters as those in the comparison experiment.
Results

The network was trained for 300 epochs, where one epoch consists of presenting all input-output pairs from the training data set once. The training error $E_{tr}$ is defined as

$$E_{tr} = \sum_{i=1}^{L} \left( (t_{N,i} - o_{N,i})^2 + (t_{Y,i} - o_{Y,i})^2 \right)$$  \hspace{1cm} (3.13)

Where, $L$ is the sum of the number of items in each of the 100 solutions in the training data set, $t_{N,i}$, $o_{N,i}$, $t_{Y,i}$, and $o_{Y,i}$ are the target and the output values for the $N$ and $Y$ outputs respectively while item $i$ was presented as current item to the network.

Fig. 3.10 shows a plot of the training error $E_{tr}$ for each epoch. The error decreases to 0 early in the training process and stays 0 until the end.

The network was saved at regular intervals during the training process. These saved networks were then tested with network inputs generated by the same program that was used to produce the packing problem solutions for the training data set. A testing run
Figure 3.11: Performance index. a) Sum of the size of the packed items. b) Sum of the value of the packed items.
for one problem instance continued until the container was filled up or a predetermined maximum number of trials (500) was reached. Each network was used to solve 100 problem instances, so that each obtained testing data set had the same size as the training data set.

A plot of the performance index computed over the solutions from the training and testing data sets is shown in Fig. 3.11. For each solved problem instance, we computed the sum of the values \( S_{val} \) of all items put in the container as well as the sum of their size \( S_{size} \).

\[
S_{val} = \sum_{j=1}^{N} s_j p_{j,\text{val}} \quad S_{size} = \sum_{j=1}^{N} s_j p_{j,\text{size}}
\]  

(3.14)

Where, \( N \) is the number of all items, \( s_j \) is 1 if item \( j \) is placed in the container or 0 otherwise, and \( p_{j,\text{val}} \) and \( p_{j,\text{size}} \) are the value and the size of item \( j \).

In the two plots are shown the minimum, mean, and maximum values computed over the 100 solved problem instances. The obtained values for the training data set are plotted at epoch 0 and the values obtained from the tests are plotted at the corresponding epoch (e.g., the results of testing the network trained for 100 epochs are plotted at epoch 100). To make

Figure 3.12: Plot of the testing errors.
the comparison easier, a dotted line is plotted at the level of the worst case in the training data set. We can see that the solutions produced by the networks trained for 230 and 280 epochs have similar performance index values to that of the solutions in the training data set. Consequently, we can select one of them to be the Final NN model (Fig. 3.9).

In the general case, we have only the training data and the performance index but we do not know the initial strategy and thus we can not compute the testing errors. However, since in the performed simulation the strategy is known, we can obtain the testing errors and compare the good networks with these selected through the performance index above.

During the testing, the outputs of the network were compared to the outputs computed simultaneously by the program that implemented the initial strategy. The resulting errors $E_{tst}$ are plotted in Fig. 3.12.

$$E_{tst} = \sum_{i=1}^{M} e_i, \quad e_i = \begin{cases} 1 & \text{if } m_{N,i} \neq o_{N,i} \\ 0 & \text{otherwise} \end{cases}$$

(3.15)

Where, $M$ is the sum of the number of items presented to the network while solving each of the 100 instances, $m_{N,i}$, $o_{N,i}$, $m_{Y,i}$, and $o_{Y,i}$ are the target (computed according to the initial strategy) and the output values for the $N$ and $Y$ outputs respectively while item $i$ was presented to the network.

The horizontal axis shows how many epochs the tested network was trained. Based on the testing errors, the best among the saved networks are those trained for 230 and 280 epochs.

From the test results is evident that the neural network learned the initial strategy and generated comparable solutions. The good networks selected through the performance index were indeed the networks with small testing errors.

**Discussion**

A neural network model was trained to approximate a strategy for solving two dimensional packing problem. In the trained model, the learned strategy was encoded as an activation pattern maintained in the PFC layer. Then, the model was used to generate solutions with similar quality according to the performance index.
Among the design choices that had to be made are the items’ features presented as inputs to the network and the number of active neurons in the hidden and \textit{PFC} layers. The problem of selecting appropriate features is very important and we intend to explore it at later stages when the core part of the current neural network model is stabilised. The reason is the general interdependency between the \textit{PFC} layer activations (strategy) and the features; the strategy is not only dependent on the features but also influences the formation of new features. For the particular simulation presented here, we chose features that simplify the learning problem.

Without a priori knowledge about the strategy that has to be learned, a simple and straightforward approach to decide the number of active neurons in the hidden and \textit{PFC} layers is to start with one and increase if the results are not satisfactory. This number is related to the number of features used in the initial strategy (in the simulation it was one feature). In this way, we start with looking for simple strategies and gradually increase the complexity. For the neural network structure used here (namely, each feature is represented with one neuron and there are one-to-one connections between the hidden layer and \textit{PFC} stripes), too many allowed active neurons lead to more false positive errors (accepting items that should be rejected), while too few – lead to more false negative errors (rejecting items that should be accepted).

Another problem that had to be addressed was the stopping criteria for the training process. We restricted the training data to positive examples only, since it is easier to extract data in this form from solved problem instances. As a consequence, the training error was insufficient to determine when to stop the training and we used an additional step consisting of testing the networks saved at different stages of the training process. To evaluate the saved networks we used the initial performance index.

For the strategy used in the simulation example, a feed-forward neural network model could be used instead of the PBWM model. We decided to use the PBWM model because it has the potential to handle more complex strategies. Future work includes the case when the initial strategy is changing over time. It can be viewed as a set of basic strategies (as the one used in the simulation) with the control switching among them and thus requires neural network model capable of expressing that dynamics.
Chapter 4

Method for Discovering Algorithms Supporting Human Skill Acquisition Abilities

4.1 Rule Embedding Approach

4.1.1 Introduction

The notion of intelligence has been influenced by the pursuit of intelligent robots. The idea of general intelligence based on the symbolic approach [36] led to powerful planners and cognitive models (e.g., Sensory Graphplan [55] and ACT-R [2]). The difficulties in applying this approach to robots resulted in shifting the focus to less sophisticated control systems like the subsumption architecture [5] which promotes the idea of simple agents operating in real time in complex environments. The intelligence of such agents stems from the interaction between the agent and the environment. This change brought robots like Roomba \(^1\) that perform useful tasks well and can be operated like other appliances, without specialised knowledge.

In other words, instead of recreating human intelligence as a whole, which is a very difficult problem, a more viable approach toward useful intelligent robots is to target only

\(^1\)iRobot Corporation
these aspects of the intelligence that are relevant to a desired task or set of tasks.

A long-term goal of our research is to create useful intelligent robots like Roomba. To look for possible mechanisms supporting the needed intelligence, we turn toward skilled humans. Considering skills as an expression of human intelligence, we search for candidate mechanisms and processes by analysing the performance of skilled individuals.

In particular, we are interested in the human abilities that support skill acquisition. In a previous work [33, 52], we proposed a testing device to investigate the ability to utilise beneficial environment features to improve task performance.

In the case of real life skills, what we can learn by observing a skilled person is limited by the fact that we have to find out simultaneously what are the various skill components and what processes support them. We can avoid this problem by using ’artificially created skills’ for which we know much more details. In this way, by observing carefully the skill acquisition process and attempts to transfer that skill to other subjects we would be in a better position to understand the involved mechanisms. This approach would be helpful in exploring the implicit skill components, i.e., the ones that the skilled person has difficulties to explain because often these components rely on processes occurring below the conscious experience level.

The next section gives an outline of the proposed strategy of skill transfer observation based on rule embedding (STORE). Then, as a specific example of this strategy, an experiment using artificial grammar learning (AGL) task is presented. The last section concludes with comments on the presented results and their relation to the long-term goal of our research.

4.1.2 Skill Transfer Observation Based On Rule Embedding

The outline of the proposed strategy for investigating human skill acquisition related abilities is given in Fig. 4.1. The three main phases are as follows.

In the first (Rule Implementation) phase, the experimenter decides on a rule that serves as a base for skillful behaviour and implements a specific task based on that rule. This task can be performed successfully of one knows the underlying rule. Furthermore, the rule can be learned by performing the task repeatedly, receiving appropriate feedback in the process.
The task and the feedback should allow for acquiring the skill in a reasonable time period. Here, we consider mainly skills that can be defined as consistently good performance of the representative task.

In the second (*Rule Embedding*) phase, a subject performs the task until some pre-defined level of performance is achieved. The task difficulty and the conditions during the task performance should be chosen so that the subject doesn’t rely exclusively on explicit learning strategies. For example, the subject may not be allowed to take notes or have off-line access to performance statistics, the reaction time may be restricted, etc.

In the final (*Rule Transfer Observation*) phase, the subject that has acquired the task related skill is asked to transfer that skill to another, naïve subject. Since the experimenter knows the initial rule that stands behind the skill being transferred, an analysis of the interaction between the two subjects would show what has been learned by the skilled subject, in what form, and possibly would suggest the supporting mechanisms.

### 4.1.3 Artificial Grammar Embedding And Transfer Experiment

In this experiment, we use an AGL task to create a situation in which we can observe the behaviour of a skilled subject while knowing the underlying rules supporting that skill.

Initially, a subject acquires the skill of recognising grammatically correct strings from incorrect ones. Later, we observe an attempt made by the skilled subject to transfer this skill to another, naïve subject.

The aim of this experiment is to test whether the subject will be able to learn to recognise correctly the grammatical strings and to analyse which parts of the underlying grammar has been learned, considering both explicit and implicit forms of the acquired knowledge.

**Artificial Grammar Embedding**

In the AGL task (an overview is given by Cleeremans [11]), the subjects are shown examples of strings, correctly formed according to some grammar, and then are asked to classify a set of testing strings as grammatical or ungrammatical. The testing set includes some of the example strings, new correctly formed strings, and ungrammatical strings.

Table 4.1 contains examples of grammatical and ungrammatical strings used to teach
Figure 4.1: Outline of the skill transfer observation based on rule embedding strategy.
Table 4.1: Examples of grammatical and test strings.

<table>
<thead>
<tr>
<th>Test string</th>
<th>Correct?</th>
<th>Test string</th>
<th>Correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSSXXKPS</td>
<td>O</td>
<td>PKK</td>
<td>O</td>
</tr>
<tr>
<td>TXXKPXKK</td>
<td>X</td>
<td>PSSSSXS</td>
<td>X</td>
</tr>
<tr>
<td>TPXTTTKK</td>
<td>?</td>
<td>PSSXXKPXTKPS</td>
<td>?</td>
</tr>
<tr>
<td>PTTKPXKK</td>
<td>?</td>
<td>TSSSSKS</td>
<td>?</td>
</tr>
<tr>
<td>PTTKPXKS</td>
<td>?</td>
<td>TKPXTTTTTTTK</td>
<td>?</td>
</tr>
<tr>
<td>TSSPXKPS</td>
<td>?</td>
<td>TSXXTKPXTKPS</td>
<td>?</td>
</tr>
<tr>
<td>PKPXTKPS</td>
<td>?</td>
<td>PTTKPS</td>
<td>?</td>
</tr>
<tr>
<td>PXPXTXPS</td>
<td>?</td>
<td>TPS</td>
<td>?</td>
</tr>
</tbody>
</table>

the grammar to the subject. Trying to guess some of the test strings before looking at the underlying grammar gives some perspective of how difficult is the task faced by the subject.

Following the outline shown in Fig. 4.1, in the first step the experimenter designs the grammar rules and the corresponding task in which the subject has to discern grammatical from ungrammatical strings.

In the rule embedding step, the subject acquires the desired skill by repeatedly performing the classification task, receiving the correct answers as feedback. When the subject reaches a predefined, desirable level of performance and is able to maintain it, we may conclude that he has internalised in some form the grammar designed by the experimenter and the rule embedding was successful.

A Finite State Automaton (FSA) describing the correct grammar is shown in Fig. 4.2. To obtain ungrammatical strings, five variants of the initial FSA (shown in Fig. 4.3) were created by swapping the positions of two characters. The swapped pair of characters were chosen so that the set of strings from the newly created FSA do not overlap with the set of strings from the initial FSA. Each ungrammatical string is created by randomly selecting one of the FSA in Fig. 4.3 and then generating a string with it. The minimum length of the strings used in the test was 3 characters and the maximum length was set to 18 characters.

In Fig. 4.4, \( A \) represents the set of all strings that can be obtained from the characters \( K,P,S,T,X \). The strings following the grammar defined by the FSA in Fig. 4.2 form the
set of grammatical strings $G \subset A$. This set in turn is divided in two subsets $G_1$ and $G_2$ such that $G_1 \cap G_2 = \emptyset$ and $G = G_1 \cup G_2$. Subset $G_1$ is used to teach the grammar to the subject. Subset $G_2$ is used to check whether the subject has learned the underlying grammar structure or just relies on exactly remembered correct strings.

At the beginning of the experiment, the subject was shown a sequence of 30 grammatical strings with each string displayed for 1 second. Then, the subject was tested on sets of 50 strings generated dynamically, with equal probability of showing grammatical or ungrammatical string. Each string in the set was displayed until the subject made a guess, followed by a feedback indicating the correctness of the guess. The immediate feedback allowed the subject to improve his guesses during the test. The subject was not allowed to take notes during or after the tests. Information about the grammar could be obtained only by performing the task.

In the next part of the embedding experiment, the grammatical strings shown to the subject were taken from the subset $G_2$, i.e., strings that have not been shown before. Similarly, the shown ungrammatical strings were also new to the subject. The purpose was to exclude the possibility of the subject recognising and using exactly remembered whole strings.
Figure 4.3: FSA for generating ungrammatical strings.
Transfer of the Embedded Artificial Grammar

Here, the subject from the above experiment attempts to transfer the newly acquired skill to another, naïve subject. An outline is shown in Rule Transfer Observation part of Fig. 4.1. Since we have detailed knowledge about the rules necessary to support the skill (i.e., the grammar defined by the FSA in Fig. 4.2, the interaction between the two subjects during the transfer may reveal which parts of the original grammar were acquired explicitly. Also, it is important to look for attempts made by the skilled subject to rationalise or give some account for the implicit components, i.e., the cases when there is no explicit rule but the subject relies on some sort of hunch or intuition.

In the transfer experiment, the communication between the skilled subject S1 and the naïve subject S2 is constrained to writing in a common protocol. First an example string is presented, subject S1 makes a guess and writes a short comment explaining the reason for the particular guess. Then, the correct answer is given by the experimenter, after which subject S1 may add some final comments for this trial. If there are any questions from subject S2 concerning the comments made by subject S1, they are written with the corresponding answers in the protocol.

The strings used in the transfer attempt were also new to the skilled subject S1, i.e., not shown in the previous experiment.

Experiment Results

A plot of the performance of the subject in the artificial grammar embedding phase is shown in Fig. 4.5. The horizontal axis shows the sequential number of the presented string set. The
The grammatical strings in the first 75 sets were chosen from the the subset $G_1$ (Fig. 4.4). After about 50 sets, the subject achieved and was able to maintain high proportion of correct answers. The relatively good performance in the first few sets can be explained by the fact that they were immediately following the example correct strings. The probability of generating short strings (e.g., with 3 to 5 characters) is higher than that of the long strings and also the number of possible short strings is small. Thus, in the initial 50 trials the subject was often tested with repeated short strings. The slight drop of performance about...
set 50 is a result of presenting only strings longer than 7 characters in the consequent tests, eliminating in this way the influence of the remembered short strings. Nevertheless, the subject was able to maintain about 80 percents correct answers. These results suggest that the subject has internalised the grammar in some form or other.

To confirm this, we continued the experiment with test strings chosen from the subset $G_2$. The results are shown in the last 10 sets starting from set 75, labelled as embedding test in Fig. 4.5. Here, although we were using test strings not shown in the previous phase, i.e. new to the subject, there was no drop in the performance. Even more, the proportion of correct answers continued to increase. From this, we can infer that the subject has successfully captured essential regularities in the presented grammatical strings and not just memorising the correct strings as a whole. Consequently, the artificial grammar was embedded successfully and the skill related to recognising the language was acquired by the subject.

Distinguishing grammatical strings is an easy task if we know the explicit grammar rules. This subject achieved something much more difficult. From examples of grammatical and ungrammatical strings, he was able to grasp the underlying regularities in a suitable form, which allowed him to generalise to previously unseen example strings. We are interested in this ability to capture regularities in its relation to human skill acquisition.

Excerpts from the protocol of the transfer attempt are shown in Table 4.2, Table 4.4, and Table 4.5.

The first thing that can be noted is that the subject decomposes some of the presented test string into substrings, without being instructed to do so, and bases his decision on the feeling of correctness of the individual substrings.

Even though the subject does not know the structure of the FSA, he chooses very reasonable places to separate the strings. After analysis based on the characters immediately after the separation, it turns out that the separation point is relatively consistent and coincides with state 4 of the FSA (see Table 4.2 and Fig. 4.2).

This separation point breaks the loop formed by states 2, 3, and 4. Except for the initial and final substrings, the correct substrings start from state 4 and end in state 4. In this way, the order among them is not important, which leads to reduced amount of information that has to be considered and simplified decision process. The other states in this loop are more
complex if we consider the state’s inputs and outputs. While state 4 has only one input, the state 2 has two inputs and a loop, and state 3 has two inputs.

The subject has learned some explicit rules about the beginning or ending of the strings as can be seen from comments 3, 10, and 12 in Table 4.4. On the other hand, the decomposition of the test string is based on intuitive feeling of ‘rightness’ of the resulting substrings as seen from comments 4, 6 and 11 in Table 4.4 and the answer 4 in Table 4.5. There is one more explicit rule. According to comment 2 in Table 4.4, the order among the substrings is not important.

The subject also noticed that repetitions of S and T, corresponding to the loops at states 1 and 2 of the FSA, are not directly related to the correctness of the strings (e.g., comment 2 from Table 4.4).

4.1.4 Discussion

In this chapter we proposed a strategy for skill acquisition analysis. The underlying idea is to create a situation where we know the rule that supports given skill and we can observe the performance of the skilled subject during skill transfer. In the presented experiment the rule was an artificial grammar and the skill was to distinguish grammatical from ungrammatical strings.

We used AGL because it is a standard tasks in Implicit Learning research and allows for investigating learning processes that occur below conscious experience level.

Usually, the AGL task is used with short learning periods and the goal is not to make the subject to learn the whole grammar, but to investigate the dissociation between performance and conscious knowledge, or in other words, between the number of correctly recognised strings and what the subject reports to know about the grammar. Here, we wanted to achieve rule embedding and continued the training until the subject attained about 80 percents correct answers and was able to maintain this level of performance.

When the subject was able to classify the test strings significantly above the chance-level, we were interested in what actually has been learned, which parts of the acquired
knowledge could be explained by the subject and which couldn’t. The form of the knowledge acquired by tested individuals has been subject of various research. The results reported by Perruchet [42] are related to our results, in the sense that some decisions on the test strings’ correctness are based not on the strings as a whole neither on individual characters but on some intermediate level. Perruchet presented results from testing subjects on pairs of consequent characters to support the idea that the subjects learn explicitly such pairs from the presented grammatical strings.

In the presented experiment, the number of strings and their average length are greater than these used by Perruchet [42], allowing in this way the subject to learn longer substrings. In the same time, these substrings are not explicitly recognised. As seen from the comments during the skill transfer attempt in Table 4.4 and Table 4.5, the subject relies on an intuitive feeling of correctness of the resulting substrings. If we consider only the grammatical strings as described by the FSA in Fig. 4.2, separating the test strings always between K and P seems to be a good explicit rule. But as can be seen in Fig. 4.3 there are three automata for ungrammatical strings that allow this decomposition too. Consequently, this explicit rule is not appropriate.

To summarise, the skilled subject in the presented experiment used both implicit recognition and explicit rules. Implicit recognition was used for deciding which are the intermediate substrings. The explicit rules were about characters in the beginning and in the end of some test strings, and the repetition of some individual characters. It is important to note one more explicit rule that becomes possible due to the implicit recognition of substrings. This is the rule stated in comment 2 in Table 4.4, namely, that the order among the substrings is not important.

We believe that the observed subject’s ability to form implicitly abstractions over basic inputs and later to build explicit rules based on these abstractions plays indispensable role in the skill acquisition process. In our research we want to find more such abilities and by understanding the mechanisms that support them to build devices that enhance the human abilities for skill acquisition. Such fundamental mechanisms would provide also support for useful intelligent robots. For that purpose, we are planning further experiments using the proposed strategy with other tasks and different sensory modalities.
Table 4.2: Decompositions of the test strings proposed by the skilled subject. The numbers at the split are added later to show the corresponding correct FSA states where possible.

<table>
<thead>
<tr>
<th>Trial No</th>
<th>Test string</th>
<th>Guess/Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PTTTTTTKPTTTKS</td>
<td>×/×</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>PTTKPKPXTTTTPKXKK</td>
<td>○/○</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>PTTK⁺</td>
<td>PXK⁺</td>
<td>PXTTTPKX⁺</td>
</tr>
<tr>
<td>3</td>
<td>TSSSXXTTTTTTSKS</td>
<td>×/×</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>TXXKPKPXTKPKPTKPS</td>
<td>○/○</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>TXXK⁺</td>
<td>PXK⁺</td>
<td>PXT⁺</td>
</tr>
<tr>
<td>5</td>
<td>TSSSXXKPXTKPKXKK</td>
<td>○/○</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>TSSSXXK⁺</td>
<td>PXK⁺</td>
<td>PXK⁺</td>
</tr>
<tr>
<td>6</td>
<td>TSSKXTTTTTXK</td>
<td>○/×</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>PTPXTPTTTTPXPS</td>
<td>○/×</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>PTT⁺</td>
<td>PX⁺</td>
<td>PXTTT⁺</td>
</tr>
<tr>
<td>8</td>
<td>TSSSSSSSPXTKPKXKK</td>
<td>○/×</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>TSSSSSSSS⁺</td>
<td>PXK⁺</td>
<td>PXK⁺</td>
</tr>
<tr>
<td>9</td>
<td>TSXXTTTKPKXPTTTK</td>
<td>○/○</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>TSXXTTTK⁺</td>
<td>PXK⁺</td>
<td>PXT⁺</td>
</tr>
<tr>
<td>10</td>
<td>PSSXXKPXTTTTKK</td>
<td>×/×</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>TKXXTXPTTTTPXXPS</td>
<td>×/×</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>TSXXTTTTKPS</td>
<td>○/○</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>PTTTTTTTTTTPXKKK</td>
<td>○/○</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>PTTTTTTTTTTTPX⁺</td>
<td>PXKX⁺</td>
<td></td>
</tr>
</tbody>
</table>

Continue...
Table 4.3: Continuation of Table 4.2.

<table>
<thead>
<tr>
<th>Trial No</th>
<th>Test string</th>
<th>Guess/Score</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>TSSKXXPXXPXTPXXPS TSSKXX</td>
<td>^PXX</td>
<td>^PXX^PXX</td>
</tr>
<tr>
<td>15</td>
<td>PTTTTTTTTTTTTTK</td>
<td>□/□</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>TSSSSSKXXPXXPXXTXTXK</td>
<td>×/×</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>PTTTTTXTTTTTTTTTTK PTTT</td>
<td>^PXXTTTTTTTTTK</td>
<td>□/□</td>
</tr>
<tr>
<td>18</td>
<td>TXXTXTTTTKPXKPTKK TXXTXTTTK</td>
<td>^PXX</td>
<td>^PXX</td>
</tr>
<tr>
<td>19</td>
<td>XTTTPXTTTPXKXXKK</td>
<td>×/×</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>PTKPXTTTTKPXKPTTTTKK PTK</td>
<td>^PXX</td>
<td>^PXX</td>
</tr>
<tr>
<td>21</td>
<td>TTTKPKPXKPPXTTTTKK TTK</td>
<td>^PXX</td>
<td>^PXX</td>
</tr>
<tr>
<td>22</td>
<td>TSSKXXPXXPXXTXTXK TSSKXX</td>
<td>^PXX</td>
<td>^PXX</td>
</tr>
<tr>
<td>23</td>
<td>TSXXKPXTTTTKPXKPS TSXXK</td>
<td>^PXX</td>
<td>^PXX</td>
</tr>
<tr>
<td>24</td>
<td>TSXXTKPTTTKPKXTTKK TSXXTK</td>
<td>^PXX</td>
<td>^PXX</td>
</tr>
<tr>
<td>25</td>
<td>PTKPXTKPXTKPKXKS PTK</td>
<td>^PXX</td>
<td>^PXX</td>
</tr>
<tr>
<td></td>
<td>Total score</td>
<td>20/25</td>
<td></td>
</tr>
</tbody>
</table>

82
Table 4.4: Some comments made by the skilled subject in the transfer experiment protocol.

<table>
<thead>
<tr>
<th>N</th>
<th>Comments</th>
<th>Trial No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The overall form. One word (substring) is a combination of several characters. I may be wrong but I would say that the last 3 characters are different*. TKS is wrong. Up to the middle it feels right.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>The order is not important. The separated parts are correct.</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>TKS is different. TKS can be separated but the rest cannot.</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>The overall feeling. There are many patterns with a sequence of S’s after T. When all separate parts are good, the whole string also feels correct.</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Feels correct but the first P seems to be different. The pattern PSS seems to be different.</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>The string as a whole feels different. In most cases, when the string is different, I don’t know where to separate it.</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>The whole string is correct. In this case, the whole string feels like one word, there is no separation.</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>The separate words seem to be wrong.</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>The whole string is one word.</td>
<td>15</td>
</tr>
<tr>
<td>10</td>
<td>The strings beginning with X are wrong.</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>The separate parts seem to be correct.</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>The last TXK is different. The word in the middle of the string also is slightly different.</td>
<td>22</td>
</tr>
</tbody>
</table>

*different is used in the sense of wrong, ungrammatical.
Table 4.5: Some questions by the naïve subject (Q:) and the corresponding answers by the skilled subject (A:).

<table>
<thead>
<tr>
<th>N</th>
<th>Questions and Answers</th>
<th>Trial No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Q: Isn’t KPX wrong?</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>A: The string can be separated between K and P and also between T and T.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Q: Do you decide based on the form (as in picture)?</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>A: That might be possible, a little bit. Here, S is not a problem.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Q: If a string cannot be separated, does it mean that this string is ungrammatical?</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>A: Can be separated = The separate words are correct.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Q: Do you remember the characters in a row?</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>A: I remember the overall feeling of the correct words.</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

Method for Discovering Beneficial Environmental Features Utilised by Skilled Humans

5.1 A Testing Device for the Human Ability to Utilise Beneficial Environmental Features

5.1.1 Introduction

Many high quality products and services rely on the availability of skilled professionals. The competitiveness and the success of product or service companies critically depend on their ability to find skilled employees.

One approach to alleviate this problem is to develop robotic devices that can perform not only simple, well defined tasks but also tasks requiring human skills (Fig. 5.1 a). A common strategy, taken in the research on skillful robots, is to analyse expert’s performance in representative tasks, to extract essential skill components, and to implement them in a robotic device. For example, Iida et al. [27] apply this approach to assembly-line pick-and-place task.

Another line of research, addressing the dependency on skilled personnel, is directed
toward providing more efficient training process. Yokokohji [57] and Nechyba [34] argue for the use of coaching devices to facilitate the skill transfer from a skilled person to a novice (Fig. 5.1, b). Detailed results for skill transfer in a pole balancing task are presented by Fujimoto et al. [24].

A common difficulty faced by both approaches is identifying the relevant skill components in the skill analysis step. While the necessary actions may be induced from the task requirements, finding on what information the skilled person bases his decisions to take one action or another is much more difficult. Taking into account the vast amount of raw information that we receive through our senses, and the need to consider also possible correlations in time scale, gives an idea why these approaches have shown limited results for complex skills.

Unlike the two approaches mentioned above, here we propose a different device use. The purpose of this research is to devise not a skillful or a coaching device but a device for testing human skill acquisition potential (Fig. 5.1 c). In other words, the device is used not to replace or train but to identify prospective employees with skill acquisition potential.
Various tests, like SPI (Synthetic Personality Inventory) \(^1\), etc., have been used for assessing general personality aspects of potential employees. In addition to general cognitive abilities, Ackerman [1] investigated also the relation between perceptual speed and psychomotor abilities, and performance in Air Traffic Control task, which is representative of the class of short term skill acquisition tasks.

However, compared to the general abilities mentioned above, abilities that are more closely related to the skill acquisition process itself have not received enough attention. Here we propose a device for testing the human ability to utilise, even unconsciously, task relevant beneficial environmental features for performance improvement.

A specific implementation of the proposed device is presented in the next section, together with an example illustrating the testing process. A discussion of the idea behind the proposed device and its relation to the human skill acquisition process follows in the third section. The last section concludes with comments on the current results and directions for further development.

### 5.1.2 Testing Device and Illustrative Example

Here, we describe a specific implementation of the proposed testing device based on an alternative choice task, and present an illustrative example of the test. First, the device and the task are described as seen from the tested subject’s point of view. Then, follows an explanation of the behind-the-scene part that shows the actual idea of the test. Finally, we give an example illustrating the testing process and interpretation of the results.

**Device and task from the subject’s point of view**

**Device**  
Fig. 5.2 shows the configuration of the proposed device as seen by the tested subject. The device consists of a computer with a program implementation of an alternative choice task, a visual display (monitor) and a keyboard.

**Task**  
The task that have to be performed is Bounce-or-not task, in which one of two possible motion patterns is presented on the visual display. The subject is asked to guess

\(^1\)http://www.recruit-ms.co.jp
which pattern was shown and to reply with a key-press.

The two possible motion patterns, Bouncing and Not Bouncing, are shown in the left part of Fig 5.3. Each motion pattern is illustrated with drawings of the positions of the moving objects.

In the top row, at time $t_0$, two discs are shown in the upper corners of the screen. Then, these two discs begin to move at the same speed along the diagonal lines toward the opposite corners. The numbers and arrows are added here only to illustrate the discs’ trajectories and are not shown to the subject. In the middle row, at time $t_c$, the discs overlap at the centre of the screen. In the bottom row, at time $t_f$, the final positions of the discs are different for the two motion patterns. In the Bouncing pattern, after bouncing off one another at time $t_c$, the discs end in the corners on the same side of the screen as their initial positions. In the Not Bouncing pattern, the discs stream through at time $t_c$ and end in the corners on the opposite side of the screen.

The two discs are displayed in colour. When the difference between the discs’ colours is large, as shown in the two patterns on the left in Fig. 5.3, distinguishing bouncing from not bouncing is easy. The actual patterns presented in the test, as the one illustrated in the right half of Fig. 5.3, use small distance between the disc’s colours, making the task much more challenging.

**Procedure** During the test, randomly selected motion patterns are presented in a sequence and the subject is asked to guess correctly as many as possible. At the end of each sequence, the proportion of correct answers is calculated and shown to the subject.

**Behind-the-scene part of the device and the task**

**Device** Fig. 5.4 shows the full configuration of the testing device. This configuration includes in addition an audio display (speakers) that is placed away from the subject’s view. The subject is not informed about the speakers and their role in the test.

**Task** The Bounce-or-not task also contains details that are hidden from the subject. As shown in Fig. 5.5, besides the motion patterns presented on the display, for some patterns a barely audible beep sound is introduced at time $t_c$ without informing the subject. Thus,
task relevant information is communicated to the subject in two ways: openly and subtly. The openly provided information is presented through the visual display, and the subtly introduced information is presented through the audio display.

**Procedure** The actual idea is to test whether or not the subject will notice the subtly introduced beep sound among the various other background stimuli coming from the environment and utilise it to improve the task performance.

Initially, the subject is tested on the motion patterns without using the beep sound until the task performance is stabilised, i.e., three consecutive performance measurements are within certain small interval. Then, without informing the subject, a beep sound is added only when the discs bounce and again the test continue until the performance is stabilised.

The testing device uses a t-test to compare these two task performances. A statistically significant improvement of the performance after adding the sound compared to that before the sound leads to the assumption that the subject has used the additional sound information. To confirm this assumption, again without informing the subject, the test continues with a beep sound only when the discs don’t bounce. If this results in task performance significantly below the chance level, then the assumption that the subject has used the beep
Figure 5.3: Explanation of the Bounce-or-not task, given to the subject. The two possible motion patterns, shown on the left, are *Bouncing* and *Not Bouncing*. A typical test pattern uses similar colours for the discs, as shown on the right. The task is to recognise whether the discs in the test patterns have bounced or not.
sound information is correct. In this case, the testing device accepts the subject, i.e., classifies the subject as having shown ability to utilise beneficial environmental features without being explicitly instructed to do so. Such subject is considered as having skill acquisition potential.

On the other hand, if the subject’s performance doesn’t improve after adding a beep sound when the discs bounce, the conclusion is only that the subject didn’t show the above ability and not that the subject doesn’t possess it. In this case, the testing device rejects the subject.

**Illustrative example**

**Test parameters** The colour of the disc starting from the top right corner was held constant yellow with RGB value of (255/255/0). The colour of the other disc was varied between RGB (250/250/0) and RGB (210/210/0) by changing the R and G components in steps of 10. Consequently, five levels of task difficulty were obtained. The colour distances corresponding to each difficulty level were $d = (0.02, 0.06, 0.1, 0.14, 0.18)$ after normalisation.

The significance level of the t-test was set to $P = 0.01$.

One female and two male subjects in their early 20’s participated in the test. The tested
Figure 5.5: In some trials, an additional beep sound is added to the Bounce-or-not task at time $t_c$. 
subjects were instructed to watch the displayed motion pattern and to answer with a key-press whether the discs had bounced or not.

The test consisted of several sessions. Each session consisted of 10 blocks of trials. In one block, 60 or 30 trials were performed sequentially, where displaying one motion pattern and receiving an answer is counted as one trial.

In each block of trials, the task difficulty was held constant. Each session started with the easiest task level, then the task difficulty was increased gradually to the maximum level and decreased again to the minimum.

The proportion of correct answers was calculated for each block of trials and was shown to the subject at the end of the block.

Test results  Fig. 5.6 shows the task performance of the subject that was accepted by the testing device. The three lines in the graph represent the proportion of correct answers in the three cases of the task: when no beep sound is used (no sound), with beep sound only when the discs bounce (with sound), and with beep sound only when the discs stream through (reversed sound). The points show the mean value of the proportion of correct answers and the bars show the standard deviation. The horizontal axis shows the distance $d$ between the colours of the two discs, which corresponds to the task easiness.

Compared to the case without sound, in the case in which similarly to the physical phenomenon there is a beep sound when the discs bounce, the proportion of correct answers significantly ($P < 0.01$) increased.

This suggests that the subject has used the beep sound information even though it was not explicitly mentioned in the task instructions and was introduced without informing the subject. In other words, this is an example of the skill related ability to utilise beneficial environmental features to improve the task performance.

In the case in which there is a beep sound only when the discs stream through, for the most difficult level $d = 0.02$, the proportion of correct answers is significantly ($P < 0.003$) below the chance level. This result confirms that the performance improvement in the previous case is due to the beep sound information.

The test settings are such that if the subject consciously perceives the sound and relies on it, the maximum proportion of correct answers can be achieved. But from Fig 5.6 we
can see that the subject’s performance in the no sound case is not the maximum possible. This and the below chance-level performance mentioned above suggest that the beep sound information may have been used at a level below the conscious experience.

Fig. 5.7 shows the task performance of the subjects that were rejected by the testing device. For these two subjects, there are no significant differences between the proportion of correct answers in the two cases: without beep sound and with beep sound when the discs bounce. The reversed sound case didn’t show significant decrease in the task performance either. These results suggest that the subjects didn’t use the beep sound information in their decisions.

We can see from Fig. 5.6 and Fig. 5.7 that one of the rejected subjects has shown similar performance level to that of the accepted subject for colour distance \( d \geq 0.06 \) in the with sound case. If the testing device was based only on the absolute task performance, both subjects should have been accepted. But since we are interested in the skill acquisition
Figure 5.7: Task performance of the subjects that failed the test, i.e., were rejected by the testing device.
Figure 5.8: Performance of the accepted (pass) and the rejected (fail) subjects in the no sound and with sound cases for colour distance $d = 0.02$.

ability of the subject, the testing device is based on detecting specific improvements of the task performance. Thus, to be accepted by the testing device, the subject has to show significant improvement of the performance in the with sound case compared to no sound case, i.e., to show ability to utilise beneficial environmental features to improve the task performance without being instructed to do so.

Fig. 5.8 gives another view of the subjects’ performance in the no sound and with sound cases for colour distance $d = 0.02$. The horizontal axis shows the number of block of trials and the vertical axis shows the proportion of correct answers in the corresponding block of trials. We can see an improvement of the accepted subject’s performance starting after 10 block of trials, while the performance of the rejected subjects doesn’t improve. The accepted subject was able to utilise the additional sound information to overcome the initial asymptotic task performance.
Generalised form of the device

The device implementation presented above was based on the Bounce-or-not task and used visual and audio stimuli. Other implementations can be obtained using different tasks and different stimuli, providing that the following essential characteristics of the device and the task are retained.

Generally, the device serves as an interface through which the subject performs a given task. There are two or more task relevant stimuli that are necessary to achieve good performance. In its role as an interface, the device gives us control over the relevant stimuli that are presented to the subject. We should be able to control which stimuli are presented and their intensity level. The device, also, computes appropriate task performance based on the presented stimuli to the subject and the subject’s response.

Using the device, we prepare three test cases: normal, enhanced and reversed. In the normal task case, only a part of the relevant stimuli are presented to the subject. In the enhanced task case we add in a subtle manner some or all of the task stimuli that were held out in the normal case. In the reversed task case the stimuli added in the enhanced case are reversed so that they would trigger incorrect responses to the task.

The testing procedure is the same as that in the illustrative example. At the beginning, the subject is tested with the normal test case until the performance stabilises. Then, without informing the subject, the test continues with the enhanced case. The performance in these two test cases is compared. If there is statistically significant improvement of the performance in the enhanced case, the test continues with the reversed case to confirm that the observed improvement is due to the use of additional information provided by the subtly introduced task stimuli. A drop in the performance after reversing these stimuli confirms that the subject indeed used them. Because of the observed sudden drop in the subject’s performance, we call this a slump-like effect. This effect is more pronounced if the subject uses the additionally introduced stimuli unconsciously.

Finally, the device accepts a subject that is able to improve his task performance by utilising these additional stimuli.

One more aspect of the task used in the test should be considered. The decision for including beep sound in the Bounce-or-not task is based on the effect of sound on visual
perception discussed by Sekuler [47] and Shimojo [54]. Due to our experience with bouncing as a physical phenomenon where bouncing is often correlated with a sound, we have already acquired the relation between sound and bouncing. This reduces the time requirements of the experiment, since the subjects don’t have to learn this relation but only to notice and use, consciously or not, the subtly introduced sound information.

5.1.3 Comments on the Idea Behind the Proposed Testing Device

Here, we discuss the idea behind the proposed testing device in relation to the implicit part of the skill acquisition process, the stepwise character of the performance improvement, and the more general difference between humans and machines.

Implicit learning

Children learn to speak their mother tongue without any formal understanding of its grammar. They can speak more or less correctly, as if they follow the language rules, while at the same time they are not aware of the rules themselves. This kind of knowledge and its acquisition are subjects of the research on implicit learning [11]. Similar observation can be made in the case of skilled practitioners. They perform very well on given tasks but sometimes have difficulties to state explicitly how they achieve this. This suggests that implicit learning is also a part of the skill acquisition process. In the proposed testing device, we address an important aspect of the learning process, namely, noticing and utilising relevant information that helps to improve the task performance. Further experiments with testing device based on different tasks are considered to investigate the implicit part of the skill acquisition process.

Skill acquisition as stepwise performance improvement

Posner [21] distinguishes “cognitive”, “associative”, and “autonomous” phases in the skill acquisition process. In the initial “cognitive” phase, the individuals acquire understanding of the task, which information is important and which is not, what actions are appropriate, etc. In the second “associative” phase, a coherent strategy and behaviour are formed, that result in acceptable level of performance. In the third “autonomous” phase, the individuals
achieve a level in which the skill performance require little or no conscious control. Usually the initial phases are associated with major performance improvements, while at the last phase, an asymptotic performance level is achieved with little further improvements.

Ericsson’s theory of expertise [20, 19, 18] explains the role of the deliberate practice in avoiding full automaticity and overcoming the asymptotic performance level. The deliberate practice is characterised with active attempts to go beyond the current performance level and often with the use of specific training situations. This results in refining the existent and forming new mental representations that improve the control over the task performance.

Based on the above positions, we view the skill acquisition process, as a repeating cycle of finding new or improved mental representations, adjusting the task related decisions and actions to utilise the new information, rehearse the newly created associations, and look again for another possibility for improvement. This can be illustrated with the curve in Fig. 5.9, showing a point of overcoming an asymptotic performance with initial accelerated performance improvement and setting for another asymptotic level. The human abilities that facilitate and maintain this process of stepwise improvements are essential for the skill acquisition.

We consider individuals as having skill acquisition potential if they are able to overcome such asymptotic performance levels on their own, i.e, without being explicitly instructed how to do so.

Based on the idea that the ability to notice and utilise beneficial environmental features
among the information coming through our senses is a general factor for task performance improvement, we can use the proposed testing device for testing skill acquisition potential.

**Intelligence in humans and devices**

If we consider using the proposed testing device to test robots instead of human subjects, an important advantage of humans comes to attention. This advantage is related to the frame problem [31], or more precisely to that, unlike robots, humans are able to overcome this problem. The frame problem arises from the need to be able to distinguish which information is important for achieving a given task and which is not. Even if we consider robots with learning capabilities, it is the humans that have to decide which sensory inputs and in what form should be fed to the learning algorithm to achieve good results in acceptable time limits.

In that context, the proposed testing device doesn’t attempt to overcome the frame problem but to test for the human abilities that allow them to overcome that problem, i.e., the abilities to find and utilise task related information that supports better performance.

**5.1.4 Discussion**

Here, we proposed a device for testing the human ability to utilise beneficial environmental features for skill improvement. We presented a concrete device implementation, based on Bounce-or-not task using combinations of visual and audio stimuli. An illustrative example of the testing process shows the device capturing successfully the ability to utilise beneficial environmental features for task performance improvement in one of the subjects. The importance of that ability for skill acquisition suggests that the proposed device can be useful for testing skill acquisition potential.

The presented device implementation shows one possible way for testing for the desired ability. Further experiments with adapting the generalised form of the proposed device to different tasks and sensory modalities are necessary to investigate the reliability and the feasibility for practical application. Variations for both cognitive and motor skills are also considered.

Another point of view allows a modification of the proposed device to be used for
obtaining information about which environmental features are used by a skilled subject in performing a given task. With a suitably selected task, ‘reversing’ features and consequent drop in performance, would reveal those used by the subject. This would be useful for unconsciously used features, that are difficult to obtain by interviewing the skilled subject.
Chapter 6

Conclusions

This thesis presented research on identifying and developing algorithms for realising robotic intelligence. We proposed two algorithms inspired by high-level human cognitive abilities, and two algorithm discovery methods targeting algorithms used in low-level, basic skill acquisition human abilities.

The first proposed algorithm, inspired by the human ability for mental rehearsal, provides functionality for fast switching among the learned behaviours in accordance to environment and task requirements. Three main aspects were considered in the implementation process of the proposed algorithm. The first one is related to the acquisition of the internal representations and their role in the input-output mapping. The second aspect is related to the use of the internal representations in the context layer to select one among several competing input-output mappings. And the third aspect is about extending the neural network structure to include a model of the environment and using the predicted sensory inputs to provide learning from imagined experience or mental rehearsal. The performance of the final implementation was tested on a simulated random walk task. Further development of the neural network algorithm implementation should be directed toward controlling robots in real-world environment.

Then, from algorithms for finding good strategies given only reward signals, we turned our attention to algorithms that would learn from existing strategies used by skilled individuals to achieve good task performance. The goal of the second proposed algorithm is to provide realisation of abilities shown by skilled humans in solving specific tasks. The
presented cognitive control based implementation of the proposed algorithm was modified to allow learning from positive examples only. The learning performance of the proposed algorithm was compared to the performance of one-class supporting vector machines and auto-encoding neural network in a test using data from human solved packing task instances. In additional test was confirmed that the algorithm can produce packing task solutions with similar performance index as the one of the solutions used for training. Further development of the algorithm implementation should be directed toward increasing the complexity of the learned strategies resulting from both, more input features and possibly supporting dynamic combinations of input features.

In both algorithm implementations discussed above, we used structure based on a human cognitive control model and searched for good parameters. On the other hand, in Chapter 4, without assuming specific structure in advance, we turn toward basic algorithms that support human skill acquisition. We proposed a method creating situations that are suitable for discovering such algorithms. These are situations in which we know the rules underlying successful task performance and can observe how a subject acquires the corresponding skill and attempts to teach it to another, naïve subject. Using the proposed method in an experiment based on artificial grammar learning task, we were able to find out a possible algorithm used by the subject to learn the implicit rules hidden behind seemingly random test strings. Further development includes implementation of the obtained algorithm for the strings domain first and attempt various adaptations to different domains like sounds, pictures, etc.

With the second algorithm discovery method, presented in Chapter 5, we assumed an algorithm with a simple input-output relation and looked for the used input features. We performed an experiment to capture the human ability to utilise, even unconsciously, beneficial environment features for improving task performance. The proposed method relied on an artificially induced slump-like effect to test for the used environmental features. Further development includes implementing the method and the corresponding test cases for different stimuli modalities.

Even though the above algorithms were presented and discussed in separate chapters, they are also closely related. The starting point of this research is implementing algorithms
Figure 6.1: A shogi example of humans successfully integrating a - mental rehearsal (e.g., imagine and select future moves), b - learning from good examples (e.g., from books with famous games), c - finding implicit regularities (e.g., specific position patterns), and d - useful environmental features, (e.g., the game partner’s facial expression), to achieve complex intelligent behaviour.
for robotic intelligence, using a neural network based cognitive control model. Both algorithms presented in Chapter 2 and Chapter 3 rely on the role of the working memory in supporting high-level human cognitive abilities like explicit learning, logical thinking, language, etc. These abilities make possible high behaviour flexibility, learning new tasks from verbal descriptions, and other characteristic human traits.

At the same time, these high-level abilities do not exist in isolation, they build on low-level information-processing abilities that support sensing, instinctive behaviours, implicit learning, etc. The algorithms targeted in Chapter 4 and Chapter 5 are related to these low-level, basic human abilities. As illustrated in a shogi example in Fig. 6.1, while performing complex tasks, humans use all of the abilities considered above. They utilise beneficial, task-related information from the environment, intuitively capture and use implicit regularities, learn from observed other humans’ task performances, and use mental rehearsal to adjust their course of actions to the arising new situations. Similarly, such integration will improve the possibilities for practical applications of the proposed algorithms. The algorithms in Chapters 2 and 3 would benefit from appropriate input abstractions reflecting implicit regularities as those considered in Chapter 4, and from discovered task relevant features as those discussed in Chapter 5. Future research pursuing not only improvement and application of the presented algorithms on their own but also focusing on the integration and interaction among these algorithms would bring better results and uncover further possible applications.
Bibliography


Appendix A

Program specifications

A.1 Packing Test

A.1.1 Purpose

This program was used for collecting data from human solved packing task instances. It presents a set of items and a container. The test subject uses the mouse to place items into the container, selecting the items according his/her own strategy. The collected solutions are saved in log files.

A.1.2 Implementation

The following platform was used for the program implementation:

Programming language: the program was implemented in the scheme programming language, using the PLT scheme environment \(^1\).

Libraries: the graphic interface library provided by PLT scheme.

Operating system: the program was developed and tested on Debian GNU/Linux.

\(^1\)http://plt.scheme.org
A.1.3 Interface

Figure A.1 shows the user interface of the program. Each item is presented with a rectangle showing its size, name, value, and weight. The container is shown on the right. Grid lines are drawn to simplify the item’s placement. The mouse is used to perform all actions. Clicking in a box containing an item selects that item. Clicking in a box containing selected item releases the selection. When an item is selected, clicking inside the container places the selected item with the lower left corner on the clicked position. If the shift key is pressed while placing the item, the item is rotated on $\pi/2$. The item is not placed in the container if there is not enough space to put it in the selected position. Once an item is placed in the container, the action can be reversed by pressing the Undo last button.

![User interface of the program for the packing task.](image)

Figure A.1: User interface of the program for the packing task.

A.1.4 Settings

Table A.1 shows the initial setting of the program. The initial settings for the items (variable items-descr-set-all) and the test item sets (variable item-sets) are given below. Each item is described by id-number, name, value, weight, width, height, and position in the container. Since the position is initially undefined, the last two numbers are set to $-1$. Each test item set is defined as a list of item id-numbers.
(define items-descr-set-all
  '( (1  "a"  0.3 3 1 1 -1 -1)
    (2  "b"  0.5 9 1 1 -1 -1)
    (3  "c"  0.9 2 1 1 -1 -1)
    (4  "d"  0.8 5 1 2 -1 -1)
    (5  "e"  0.2 7 2 1 -1 -1)
    (6  "f"  0.8 4 2 1 -1 -1)
    (7  "g"  0.4 2 2 2 -1 -1)
    (8  "h"  0.4 8 2 2 -1 -1)
    (9  "i"  0.6 8 1 3 -1 -1)
    (10 "j"  0.9 6 3 1 -1 -1)
    (11 "k"  0.2 3 1 3 -1 -1)
    (12 "l"  0.2 3 2 3 -1 -1)
    (13 "m"  0.7 8 3 2 -1 -1)
    (14 "n"  0.7 2 3 2 -1 -1)
    (15 "o"  0.6 9 3 3 -1 -1)))

(define item-sets
  ' (is1  (1 2 4 5 7 8 10 11 13 14 ))
    (is2  (1 3 4 6 7 9 10 12 13 15))
    (is3  (2 3 5 6 8 9 11 12 14 15))
    (is4  (1  4 5 6 7 8 10 11 12 15))
    (is5  (1 2 3 6 7 8 9 13 14 15))
    (is6  (1 2 3 4 5 7 9 11 13 14 ))
    (is7  (1 3 5 6 8 9 10 12 14 15))
    (is8  (1 2 3 4 6 8 10 11 12 14 ))
    (is9  (2 3 4 5 7 9 10 11 13 15))
    (is10 (1 2 4 5 6 7 8 9 12 13 ))))
Table A.1: Initial setting for the packing task program.

<table>
<thead>
<tr>
<th>Description</th>
<th>Variable</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of all items</td>
<td>—</td>
<td>15</td>
</tr>
<tr>
<td>Number of items in a test set</td>
<td>—</td>
<td>10</td>
</tr>
<tr>
<td>Container width</td>
<td><em>cont-w</em></td>
<td>5</td>
</tr>
<tr>
<td>Container height</td>
<td><em>cont-h</em></td>
<td>5</td>
</tr>
</tbody>
</table>

A.1.5 Example Session

An example session is shown in Fig. A.2. The subject selects the following items and places them in the container: ”o”, ”m”, ”g”, ”k”, ”d” and ”c”. For each item, there are two screens showing the state after selecting the item and after placing it in the container.

A.1.6 Output Data

The task solutions provided by the subject are written in a log file of the form shown below:
Figure A.2: Example session of the packing task.
The first number is a time stamp showing the beginning of the test, the second number shows the time used for solving the problem instance (in seconds), the string on the third line shows which test item set was used, the fourth line shows the number of items in the test set, the following two numbers show the width and the height of the container. Finally, there is a list of the items put in the container with the following details for each item: sequential number showing the order in which the items were put into the container; the $x$ and $y$ positions of the item’s lower left corner in the container; the item’s id-number, name, value, weight, width, and height; whether the item was rotated or not; and the item’s features max-value, min-value, max-weight, min-weight, max-size, and min-size.
A.2 Bouncing-or-not Test

A.2.1 Purpose

This program is used for testing the human ability to utilise beneficial environmental features. It presents two discs moving along the window’s diagonal lines. There are two motion patterns depending on whether the discs bounce or not when they meet at the centre of the window. The colours of the two discs are slightly different, and the subject relies on this colour difference to guess which motion pattern was presented. Later, without informing the subject, a subtle beep sound is introduced when the two discs bounce off each other. The test checks whether the subject will use the additional sound information to improve his/her guesses.

A.2.2 Implementation

The following platform was used for the program implementation:

**Programming language:** the program was implemented in the scheme programming language, using the PLT scheme environment.

**Libraries:** the graphic interface library provided by PLT scheme; the OpenGL library through the Sgl scheme extension.

**Operating system:** the program was developed and tested on Debian GNU/Linux. A version for Windows XP is available too.

A.2.3 Interface

Figure A.3 shows the user interface of the program.

The keys used in the program are described in Table A.2. During normal testing, the subject uses space key to start, and then only / and \ keys to respond whether the disks bounced or no. The rest of the keys are intended to be used by the experimenter.

---

2http://plt.scheme.org
Figure A.3: User interface of the program for the Bouncing-or-not test.
Table A.2: Description of the keys used in the Bouncing-or-not test program.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>To begin the test and to pause/continue</td>
</tr>
<tr>
<td>/</td>
<td>To indicate that the discs didn’t bounce</td>
</tr>
<tr>
<td>\</td>
<td>To indicate that the discs bounced</td>
</tr>
<tr>
<td>v</td>
<td>To change the colour of the left disc, cycling among the predefined values</td>
</tr>
<tr>
<td>b</td>
<td>To toggle the beep sound when the discs bounce</td>
</tr>
<tr>
<td>r</td>
<td>To toggle the reverse-sound case, i.e. beep sound when the discs stream through</td>
</tr>
<tr>
<td>1</td>
<td>To set the colour of the left disc.</td>
</tr>
<tr>
<td>2</td>
<td>To set the colour of the right disc.</td>
</tr>
<tr>
<td>c</td>
<td>To increase the disc size</td>
</tr>
<tr>
<td>d</td>
<td>To decrease the disc size</td>
</tr>
<tr>
<td>a</td>
<td>To increase the disc speed</td>
</tr>
<tr>
<td>z</td>
<td>To decrease the disc speed</td>
</tr>
</tbody>
</table>

A.2.4 Settings

The initial settings are shown in Table A.3. The disk speed is controlled by the waiting time between two steps (disc-dt), thus increasing this value slows down the movement of the discs.

A.2.5 Example Session

An example session is shown in Fig. A.4. The screens 1a) to 1g) present one trial. After the discs disappear at the bottom of the screen, the subject has to enter a guess by pressing / or \ key. The next trial 2a) starts immediately after the subject’s input. With the default settings, the number of correct guesses is hidden from the user, and the experimenter decides whether and when to show it to the subject.

A.2.6 Output Data

The collected data is written in a log file of the form:
Table A.3: Initial setting for the AGL test program.

<table>
<thead>
<tr>
<th>Description</th>
<th>Variable</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of trials in a set</td>
<td>max-trials</td>
<td>50</td>
</tr>
<tr>
<td>Use beep sound</td>
<td>bell?</td>
<td>false</td>
</tr>
<tr>
<td>Use reversed beep sound</td>
<td>swap-sound?</td>
<td>false</td>
</tr>
<tr>
<td>Disc size</td>
<td>disc-size</td>
<td>0.05</td>
</tr>
<tr>
<td>Disc speed</td>
<td>disc-dt</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>disc-ds</td>
<td>0.04</td>
</tr>
<tr>
<td>Colour of the right disc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Colour of the left disc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predefined colours for the left disc</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sound volume</td>
<td>sound-volume</td>
<td>25</td>
</tr>
<tr>
<td>Position of the bottom line</td>
<td>finish-line-y</td>
<td>-0.9</td>
</tr>
<tr>
<td>(the line at which the discs disappear)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure A.4: Example trial of the Bouncing-or-not test - 1a) to 1g). The second trial 2a) begins immediately after the user’s response.
<table>
<thead>
<tr>
<th>Test Case</th>
<th>Disc Speed</th>
<th>Finish Line Position</th>
<th>Colour Distance</th>
<th>Sound Volume</th>
<th>Trial Count</th>
<th>Answer Count</th>
<th>Correct Answers</th>
<th>Proportion of Correct Answers to Trials</th>
<th>Proportion of Correct Answers to Answers</th>
<th>Bounced or Not</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>ws: 12, -0.9, 0.0588, 28, 1, 1, 1.0, 1.0, 0, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ws: 12, -0.9, 0.0588, 28, 2, 2, 2, 1.0, 1.0, 1, 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ws: 12, -0.9, 0.0588, 28, 3, 3, 2, 0.67, 0.67, 1, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ws: 12, -0.9, 0.0588, 28, 4, 4, 3, 0.75, 0.75, 0, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ws: 12, -0.9, 0.0588, 28, 5, 5, 4, 0.8, 0.8, 0, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ws: 12, -0.9, 0.0588, 28, 6, 6, 5, 0.83, 0.83, 0, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ws: 12, -0.9, 0.0588, 28, 7, 7, 6, 0.86, 0.86, 1, 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ws: 12, -0.9, 0.0588, 28, 8, 8, 7, 0.88, 0.88, 0, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ws: 12, -0.9, 0.0588, 28, 9, 9, 8, 0.89, 0.89, 0, 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each line of the log file contains the following information: test case (“ns” - no beep sound, “ws” - with beep sound when the discs bounce off each other, “rs” - with beep sound when the discs stream through), disc speed, position of the finish line, colour distance, sound volume, trial count, answer count, number of correct answers, proportion of correct answers to trials, proportion of correct answers to answers, bounced or not, answer.
A.3 AGL Test

A.3.1 Purpose

This program was used in the AGL embedding test (Section 4.1). With this program, test strings are presented to the subject to classify them as grammatical or ungrammatical, the correctness of the subject answers is indicated, and the information from the trials is saved in a log file. Two types of test strings are shown to the subject: grammatical strings and ungrammatical strings. The grammatical strings are generated from the grammar we want to “embed” into the subject. The ungrammatical strings are generated from one or more grammars, making sure that they are different from the grammatical strings.

A.3.2 Implementation

The following platform was used for the program implementation:

Programming language: the program was implemented in the scheme programming language, using the PLT scheme environment ³.

Libraries: the graphic interface library provided by PLT scheme.

Operating system: the program was developed and tested on Debian GNU/Linux. A version for Windows XP is available too.

A.3.3 Interface

Figure A.5 shows the user interface of the program with labels for the main items. The test string is shown to the subject in the white field. On the left of the test string, there is an icon indicating the following: a question mark indicates that the program waits for an answer from the user; a circle indicates that the subject’s answer is correct; and an ‘x’ mark indicates that the subject’s answer is incorrect. There are three numbers in the gray field above the test string. The first number shows the number of test string sets shown so far.

³http://plt.scheme.org
Table A.4: Description of the keys used in the AGL test program.

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>To begin the test</td>
</tr>
<tr>
<td>y</td>
<td>To classify the current test string as grammatical</td>
</tr>
<tr>
<td>n</td>
<td>To classify the current test string as ungrammatical</td>
</tr>
<tr>
<td>f</td>
<td>To change the font of the test string</td>
</tr>
<tr>
<td>c</td>
<td>To change the colour of the test string</td>
</tr>
<tr>
<td>e</td>
<td>To show an example of a grammatical string</td>
</tr>
<tr>
<td>E</td>
<td>To show a set of 30 examples of grammatical strings</td>
</tr>
<tr>
<td>s</td>
<td>To show with green colour the grammatical part (prefix) of the string</td>
</tr>
</tbody>
</table>

The second number shows the number of correct answers given so far in the current set. The third number shows the number of test strings presented so far in the current set.

The keys used in the program are described in Table A.4. During normal testing, the subject uses space key to start, and then only y and n keys to classify the current test string. The rest of the keys are intended to be used by the experimenter.

A.3.4 Settings

Besides the grammar descriptions defining the grammatical and ungrammatical strings, the initial settings are given in Table A.5.

125
Table A.5: Initial setting for the AGL test program.

<table>
<thead>
<tr>
<th>Description</th>
<th>Variable</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of test sets</td>
<td><em>max-runs</em></td>
<td>5</td>
</tr>
<tr>
<td>Number of test strings in a set</td>
<td><em>max-trials</em></td>
<td>50</td>
</tr>
<tr>
<td>Minimal length of the test strings</td>
<td><em>minlen</em></td>
<td>3</td>
</tr>
<tr>
<td>Maximal length of the test strings</td>
<td><em>maxlen</em></td>
<td>18</td>
</tr>
<tr>
<td>Time interval for showing the correctness of an answer (in seconds)</td>
<td><em>fback-display-time</em></td>
<td>0.75</td>
</tr>
<tr>
<td>Log file name</td>
<td><em>log-file-name</em></td>
<td>“agl-test.log”</td>
</tr>
</tbody>
</table>

Table A.6: Description of the example test session in Fig. A.6.

<table>
<thead>
<tr>
<th>Trial number</th>
<th>Test String</th>
<th>Subject’s answer</th>
<th>Correctness of the answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TXXTKPXKPS</td>
<td>y</td>
<td>correct</td>
</tr>
<tr>
<td>2</td>
<td>TTTKPXKK</td>
<td>n</td>
<td>correct</td>
</tr>
<tr>
<td>3</td>
<td>KPXTKPXT</td>
<td>y</td>
<td>incorrect</td>
</tr>
<tr>
<td>4</td>
<td>XTTKPXKPS</td>
<td>n</td>
<td>correct</td>
</tr>
<tr>
<td>5</td>
<td>XTTKPXKPS</td>
<td>n</td>
<td>correct</td>
</tr>
<tr>
<td>6</td>
<td>KSSSXSXSX</td>
<td>y</td>
<td>incorrect</td>
</tr>
<tr>
<td>7</td>
<td>TTTKPKXKK</td>
<td>n</td>
<td>correct</td>
</tr>
</tbody>
</table>

**A.3.5 Example Session**

Figure A.6 shows an example test session. After starting the program and pressing the space key, the subject is presented with the first test string “TXXTKPXKPS”. The subject’s answer $y$ is shown in a separate box for clarity; the actual program doesn’t show it. Immediately after the subject’s answer the correctness of the answer is indicated with the icon on the left of the test string. In this case, the answer was correct as can be seen from the circle icon in the screen shown below. After predefined period of time the next test string “TTTKPKXKK” is shown. The subject’s answer $n$ is correct. The interaction shown in Fig. A.6 is summarised in Table A.6. Since the strings are generated at random, sometimes the same string is shown more that once as shown in trials 4 and 5.
Figure A.6: Example session of seven trials. Each trial is described by an initial screen with a test string, the user input, a screen showing the program’s feedback, and a trial number given on the left.
A.3.6 Output Data

Each test session creates two log files: brief and verbose. An example of the brief log file is given below.

```
1149658645  0  0
1149658744  50  40
1149658829  50  37
1149658916  50  38
1149659002  50  43
1149659089  50  45
```

```
--agl-test.log--
```

The first line indicates the start time of the program. In the rest of the lines, the first column indicates the start time time of the corresponding test set, the second column shows the number of presented test strings, and the third column shows the number of correct answers. The start times are in seconds since 1970-01-01 00:00:00 UTC.

The verbose log file is in the form shown below.

```
(1149658744
  ("PXK"  yes 0   2103.90)
  ("PTKS" no 1   1196.44)
  ("PKS" no 1   830.62)
  ...
)
...
```

```
--agl-test.logv--
```

The first number indicates the start time of the corresponding set. Then, the test string, the subject’s answer, the correctness of the answer, and the reaction time (in milliseconds) are written for each trial.
Related publications

Journals


Proceedings of International Conferences/Workshops

Full paper review


Abstract review


Proceedings of National Conferences/Workshops

