EMERGENCE AND EVOLUTION OF COMPUTATIONAL HABITATS

by

Alexander F. Tulai

A thesis submitted to the Faculty of
Graduate Studies and Research in partial
fulfillment of the requirements for the
degree of Doctor of Philosophy

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

April 19, 2004

© Copyright 2004, Alexander F. Tulai

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-97850-8

Our file Notre référence

ISBN: 0-612-97850-8

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

TABLE OF CONTENTS

<i>Chapter 1</i>	<i>1</i>
1 INTRODUCTION - COMPUTATIONAL HABITATS	1
1.1 Introduction.....	1
1.2 Motivation	2
1.3 The Contributions Made by this Thesis.....	5
1.3.1 Competitive-Cooperative Coevolution Genetic Algorithm	5
1.3.2 Parameter Free Algorithms	6
1.3.3 Manual and Automatic Speciation	6
1.3.4 Computational Habitats Theoretical Framework.....	7
1.3.5 First Stochastic Habitat Implementation	8
1.3.6 Data Classification in Computational Habitats	9
<i>Chapter 2</i>	<i>10</i>
2 STATE-OF-THE-ART LITERATURE	10
2.1 Turing and von Neumann	10
2.2 Deterministic Cellular Automata	11
2.3 Non-Uniform Cellular Automata	17
2.4 Artificial Chemistries	19
2.5 Computer Programs Worlds.....	24
2.6 Ants Colonies and Model-Based Algorithms	26
2.7 Incremental Evolution	29
2.8 Multi-Agent Systems	33
<i>Chapter 3</i>	<i>36</i>
3 FROM GENETIC ALGORITHMS TO COMPUTATIONAL HABITATS	36
3.1 Cooperation Mechanisms	36
3.2 Evolving GA Strategy Parameters	45
3.3 MSWV - Exploiting Speciation in Data Classification.....	52
3.3.1 Background.....	53
3.3.2 Algorithm Characteristics	55
3.3.2.1 Representation and Species Definition	55
3.3.2.2 Mutation Operators.....	57
3.3.2.3 Rule Discovery and Automatic Speciation.....	57

3.3.2.4	Rule Confidence and Rule Weight	59
3.3.2.5	Individual Fitness Calculation.....	60
3.3.2.6	Training Time.....	62
3.3.2.7	Testing	63
3.3.3	Experimental Results	63
3.3.4	The Importance of Speciation.....	69
3.3.5	Speed and Scalability	74
3.3.6	Final Comments about MSWV	78
3.4	The CH as an EC Model.....	79
Chapter 4.....		83
4	COMPUTATIONAL HABITATS – THEORETICAL FRAMEWORK	83
4.1	The CH-Calculus	83
4.2	Formal Representation of Stochastic Agents	87
4.3	Formal Representation of Computational Habitats.....	99
4.4	More Examples	100
Chapter 5.....		105
5	HABITAT MODEL	105
5.1	A Simple Model of Pattern Replicators	105
5.2	A Simple Model with Mutation.....	108
5.3	Doing Away with Immortality	110
5.4	Gene Exchange.....	111
5.5	The Feeder Sub-Agent	113
5.6	Autopoietic Habitats.....	121
5.7	Stochastic versus Non-stochastic Processes.....	128
5.8	Mutation versus Random Alleles Setting	132
Chapter 6.....		134
6	EMERGENCE IN STOCHASTIC HABITATS.....	134
6.1	Enhanced Stochastic Agent Model.....	134
6.2	Signaling Space Differentiation Process	140
6.3	Apoptosis and Life Expectation	143
6.4	Evolution and Adaptation in Stochastic Habitats	150
6.5	The Effect of Genetic Material Exchange	164

6.6	A Repair Mechanism in Stochastic Agents	169
Chapter 7.....		172
7	DATA CLASSIFICATION IN COMPUTATIONAL HABITATS....	172
7.1.1	Representation and Species Definition	172
7.1.2	Rule Discovery and Automatic Speciation	173
7.1.3	Training Data.....	176
7.1.4	Rule Confidence and Rule Weight	176
7.1.5	Fitness Calculation	177
7.1.6	Reward Mechanism Changes	177
7.1.7	Training Time	178
7.1.8	Testing.....	178
7.1.9	Experimental Results	178
Chapter 8.....		181
8	LONG-TERM GOALS AND CONCLUSIONS	181
8.1	Extensions to MSWV System.....	181
8.1.1	Data Driven Initialization.....	181
8.1.2	Crossover Operator.....	182
8.2	Future Research in <i>CH</i>	183
8.2.1	Simplified Model of Nature	183
8.2.2	Artificial Embryogenesis in Stochastic Agents	184
8.2.3	CH as Environments for Problem Solving.....	185
8.2.4	Theoretical Development of the CH Model.....	185
8.3	Conclusions.....	186
BIBLIOGRAPHY.....		189
APPENDIX.....		202
Tables A.1 – A.8.....		202
Implementation		213
INDEX		214

LIST OF FIGURES

Figure 1: The origins of the concept of Computational Habitats	1
Figure 2: Initial and final (dead) Langton self-replicating loop	13
Figure 3: A general cooperative coevolutionary model.....	37
Figure 4: Training patterns for the two-spiral problem.	38
Figure 5: The initial parallel GA like setup of the CCC-GA.....	40
Figure 6: The average decay of the SES as a function of the number of hidden units.....	41
Figure 7: The improvements in squared error sum (SES) after new hidden units, retraining and the sum of the two.....	43
Figure 8: The distribution of the genetic operators' probabilities for 16 sub-populations.....	46
Figure 9: Genome representation: $\{x_1, \dots, x_n\}$ are application dependent chromosomes and $\{p_c, p_m, p_r, s\}$ are chromosomes representing the GA strategy parameters.	47
Figure 10: Evolution of the population fitness for the sphere and the weighted sphere model (hyper-ellipsoid) for $n=32$	48
Figure 11: The fitness average (times 0.1) and the number of subspecies for the <i>MUS</i> dataset with speciation turned on. Testing accuracy = 99.89%.	69
Figure 12: The fitness average and the number of subspecies for the two species of the <i>MUS</i> dataset, speciation turned off. Testing acc. = 96.80%.	70
Figure 13: The fitness average and the number of subspecies for the <i>WVF</i> dataset with speciation turned on. Testing accuracy = 82.27%.	71
Figure 14: The fitness average and the number of subspecies for the <i>WVF</i> dataset with all individuals placed in 1 subspecies. Testing accuracy 70.90%.	72
Figure 15: Conceptual road from GAs to CHs.	79
Figure 16: The role of membrane protein receptors in cell signaling (The Biology Project – University of Arizona).	86
Figure 17: Stochastic agent model and cloning steps	105
Figure 18: The agent distribution in the habitat after 30 cycles.....	107
Figure 19: The agent distribution in the habitat after 100 cycles.....	108
Figure 20: Spatial distribution of agents at the end of simulation with mutation turned off...	109
Figure 21: Spatial distribution of agents at the end of simulation with mutation turned on. ...	110
Figure 22: Spatial distribution of agents at the end of simulation with mutation turned on and an apoptosis component.	111
Figure 23: Gene mixing using the transformation mechanism borrowed from prokaryotes.	113
Figure 24: The evolution of population size with 100 initial agents.	116
Figure 25: The evolution of population size with 1000 initial agents.	116

Figure 26: The evolution of population size with 1000 initial agents and a Moore neighborhood of size 1.	118
Figure 27: The evolution of population size with 100 initial agents, a Moore neighborhood of size 0 and energy conservation.	122
Figure 28: The evolution of population size with 100 initial agents.....	123
Figure 29: The evolution of population size with 1000 initial agents.....	123
Figure 30: The evolution of population size with 100 initial agents and flooding every 5000 cycles.....	124
Figure 31: The evolution of population size with 1000 initial agents and flooding every 50 cycles, energy contents [60, 80].....	126
Figure 32: The distribution of agents after 500 cycles.....	126
Figure 33: The distribution of agents after 1000 cycles.....	127
Figure 34: The distribution of non-stochastic agents after 125 cycles.	128
Figure 35: The growth of a population of non-stochastic agents.	129
Figure 36: The distribution of non-stochastic (sterile) agents after 2000 cycles.....	130
Figure 37: The growth of a (initial) population of 1000 non-stochastic agents with apoptosis probability set at 0.0084.....	132
Figure 38: The growth of a (initial) population of 1000 stochastic agents with random offspring gene setting and transformation turned off.....	133
Figure 39: Population growth with transformation turned off.....	133
Figure 40: Enhanced model of a stochastic agent.....	137
Figure 41: Life expectancy using the simple apoptosis model.....	144
Figure 42: Life expectancy using an advanced apoptosis model.....	145
Figure 43: Life expectancy according to the Gompertz law.	146
Figure 44: Life expectancy of agents using advanced apoptosis versus the Gompertz law	147
Figure 45: Population of agents with simple apoptosis after 6000 cycles.....	149
Figure 46: Population of agents with advanced apoptosis after 6000 cycles.	149
Figure 47: Agent distribution after 2000 cycles and color coding scheme.	151
Figure 48: Experiment 1: agent distribution after 3500 cycles.....	154
Figure 49: Experiment 1: agent distribution after 5000 cycles.....	155
Figure 50: Experiment 1: agent distribution after 10000 cycles.....	156
Figure 51: Experiment 1. Agent distribution after 20000 cycles	158
Figure 52: Population growth during experiment 1.....	158
Figure 53: Population growth during experiment 2.....	159
Figure 54: Population growth during experiment 3.....	160
Figure 55: Population growth during experiment 4.....	160
Figure 56: Population growth during experiment 5.....	161
Figure 57: Agent distribution and clusters probed at the end of fifth experiment	162
Figure 58: Population growth during experiment 7.....	163
Figure 59: Population growth with all mutation operators and transformation turned on.....	166
Figure 60: Agent distribution and clusters probed.....	167
Figure 61: Population growth with one gene mutator and transformation turned on.....	168

Figure 62: Life expectancy curves: with repair mechanism turned off, turned on, and the theoretical Gompertz law curve.	170
Figure 63: A theoretical distribution of agents in the rule fitness landscape.....	173

LIST OF TABLES

Table 1: A summary of known issues with classical evolutionary algorithms.....	5
Table 2: Required number of hidden units for the three methods under study.....	42
Table 3: Required number of hidden units for CCC-GA when different retraining methods are used.....	44
Table 4: Seven functions test bed.....	49
Table 5: Algorithm parameters for function optimization and CNNs evolving tasks.....	50
Table 6: Results for functions F_3 - F_7	51
Table 7: The evolved CNN length (in hidden units) for the two GAs under comparison.....	52
Table 8: The mutation operators used by MSWV.....	57
Table 9: The datasets used for experiments (unknown values given as a percentage of total).....	64
Table 10: Accuracy rates (%) of the classifiers used in the study. The row labelled W-L-T counts the wins-losses-ties of one-on-one comparison between MSWV and another classifier. The last two rows show the z-value and the confidence level of the Wilcoxon signed rank test.....	67
Table 11: Accuracy rates of the classifiers (in %) on datasets with at least one continuous numerical feature.....	68
Table 12: Accuracy rates (%) of MSWV with speciation turned on, turned off and with all individuals of a species placed initially in one subspecies.....	73
Table 13: Training time (in seconds) on 10 large datasets.....	74
Table 14: Testing time (in seconds) on 10 large datasets.....	75
Table 15: Normalized training time for MSWV and SMO in [ms].....	76
Table 16: Training time and accuracy for fractions of ADL dataset.....	77
Table 17: Summary of issues and possible solutions in the CH model.....	80
Table 18: The time (in cycles) for the agents to fill in half of the cellular space, with and without mutation.....	109
Table 19: The time (in cycles) for the agents to fill in 80% of the cellular space, with and without transformation.....	112
Table 20: The average values of the alleles calculated over the whole population at the end of simulation (except for the last three the rest of them are probabilities of running the corresponding process).....	117
Table 21: The average values of the alleles calculated over the whole population at the end of simulation (corresponding to Figure 26).....	119
Table 22: A comparison of the energy requirements for stochastic and non-stochastic agents.....	131
Table 23: The genotype of the enhanced stochastic agent.....	135
Table 24 : Enhanced model apoptosis process pseudo code.....	144

Table 25: The life expectancy of the agents in the three experiments performed.....	148
Table 26: Mutation operators used in the enhanced model.....	150
Table 27: Food replenishing schedule for the first experiment.....	151
Table 28: Experiment 1: summary of agent mutations after 20000 cycles.....	152
Table 29: Accuracy and dataset evaluations (in thousands) for MSUV, CHC and MSWV	178

ACKNOWLEDGMENTS

"Thanks are justly due for things got without purchase." - Ovid

If acknowledgments are due for things contributing to one finishing the difficult work that is a Doctor of Philosophy thesis, they surely must be also due for those things that helped one decide and begin the great journey.

No thanks and acknowledgments can be made before I thank my late father, Alexandru Tulai, who throughout his life never had a greater wish than seeing me making a contribution to science. If he had ever taught me a lesson it must be that one's work is never complete. In this spirit I see this thesis as just a step of a long-term project that I'll strive to continue. I also want to thank my mother, Georgeta Tulai, who in the best tradition of balancing and putting things in perspective, has always encouraged me to work less and find more time for myself. I think it's time for me to start listening.

I would also like to acknowledge the advice and encouragement that Edmond Nicolau, Professor of Cybernetics at University of Bucharest, has offered me together with the first opportunity to start a PhD program more than twenty years ago. The pursue of freedom has prevented me from following up on that opportunity but the regret of not being able to continue has never died.

If anybody deserves thanks for convincing me to restart my PhD journey it must be Professor Tyseer Aboulnasr who having taught me the Adaptive Digital Signal Processing course has tried to persuade me to enroll in a PhD program. I am really glad I gave in. My interest in Artificial Intelligence drove me towards Computer Science rather than Electrical Engineering but I shall always be thankful to Professor Aboulnasr for her encouragement and support.

So one day I started searching the web using the key words "Artificial Intelligence" and "Ottawa". This is how I first time got acquainted with Professor Franz Oppacher. Trying to probe the depth of my interest, Franz has wisely advised me to start by taking the Genetic Algorithms course! Which I did and the rest is history. How blessed I am, to have the insightful guidance of Professor Oppacher, who in these four years has taught me so much! Our collaboration has been rewarded by success and I hope we will have the opportunity to continue to work together.

In addition thereto, I am indebted to Professors Kranakis, Marchand, Japkowicz, Carmody and Dr. Famili, together with all my honoured Professors and teachers, as well as the ever-supportive staff of the School of Computer Science.

The family is the beginning and the end of all things so I would like to finish by expressing my thanks and gratitude for their unwavering support to my dearest friend, my wife Cristina, and my two sons, Timothy and Anthony Tulai.

GLOSSARY

AC	Artificial Chemistries
ACO	ant colony optimization
AE	artificial embryogenesis
AI	Artificial Intelligence
CA	(deterministic) Cellular Automata
CC	cooperative coevolution
CCC-GA	Competitive-Cooperative Coevolution – Genetic Algorithm
CE	cross-entropy
CH	Computational Habitat
CHAM	Chemical Abstract Machine
CHC	CH-based classifier
CNN	cascade neural network
Coreworld	A virtual world known to be computation-universal
DNA	Deoxyribonucleic Acid
EA	Evolutionary Algorithm
ECO	a general model based on eco-problem solving methodology
EDA	estimation of distribution algorithms
ES	Evolutionary Strategies
GA	Genetic Algorithm

GBML	genetics-based machine learning
GP	Genetic Programming
HU	hidden unit
IB	instance-based
ISS	internal signaling space
MAS	multi-agent systems
MBS	model-based search
m-SAT	m-bit satisfiability problem
MSUV	Multiple-Species Universal Voting
MSWV	Multiple-Species Weighted Voting
NB	Naïve Bayes
NN	Neural Network
NUCA	Non-Uniform Cellular Automata
PART	an algorithm for inferring rules by repeatedly generating partial decision trees
PCA	Partitioned Cellular Automata
PGA	Parallel GA
PROTEAN	an autopoietic ecology initialized with a population of Turing machines
P-systems	A computing model that has molecules, rules, a domain similar to a cell structure separated by membranes and an algorithm specifying how the rules are applied
Push	a LISP-like language

PushPop	an autoconstructive evolution system
RCA	Reversible Cellular Automata
RSA-SS	Reproductive sub-agent signaling space
SDSR	Structurally Dissolvable Self-Reproducing
SES	squared error sum
SGA	stochastic gradient ascent
SH	Stochastic Habitat
SMO	Sequential Minimal Optimization, an SVM classifier system
SR	self-replicating
SS	signaling space
SSA-SS	System sub-agent signaling space
SVM	Support Vector Machines
TSP	Traveling Salesman Problem
XSS	external signaling space

1 INTRODUCTION - COMPUTATIONAL HABITATS

"The beginnings of all things are small" - Cicero

1.1 Introduction

Evolutionary Computation methods have traditionally been used in two ways: as techniques for solving problems and as simplified models of nature. Our long-term goal, started in this thesis, is to tackle the fundamental task of developing habitats that not only parallel natural evolution but also solve a given problem in the process. We call these forms of evolutionary models Computational Habitats (CH). The motivation for this new approach is practical as well as methodological. As we will detail in the following chapters there are good reasons to believe this new approach will lead to new and efficient ways of solving practical problems. Figure 1 depicts the concept of Computational Habitat.

Figure 1: The origins of the concept of Computational Habitats

Although the problem has never been systematically studied, as detailed in the next chapters, various research topics are relevant and may be seen as (simplified) forms of CHs.

1.2 Motivation

Evolutionary Algorithms (EAs) are increasingly used, as alternatives to analytical methods, to solve problems made intractable by their complexity and high number of variables. However, those who have accumulated some experience in this field agree that given a potential application there is no certainty from the onset whether a GA (for example) is a good method to use or not. One needs significant domain specific knowledge to be able to navigate through the numerous decisions one needs to take. What kind of algorithm should we use? Should we use Evolutionary Strategies (ESs) or Genetic Algorithms (GAs)? What genetic operators should we use? What selection methods? What is diversity and how do we preserve it? These are only some of the questions but even after answering most of them we are not done yet because in most cases some preliminary experiments (followed by adjustments) must be performed. Why is it so difficult?

To answer this question we have to go back to the definition of what GAs really are. Genetic algorithms are search algorithms (Goldberg 1989) and, quite obvious, the outcome of any search process is fundamentally affected by how large the search space is. Users of GAs have to cope not only with the inherent dimension of the problem search space but they have also to cope with the size of the design search space as well. And not only that, almost always the search for a complex solution is done by starting with a population of randomly initialized solutions of equal dimension. Forget about designing a human, what if one would try to take on the task of designing a fly given the right numbers of cells. What is the chance to putting them all together in the right configuration? The problem actually is not new. Biologists have puzzled and argued over the possibility that proteins (for example) could have been discovered by natural selection given the size of the **protein space** determined by the number of amino-acids. If nature had to do a massive random search it would've faced the kind of problem noticed by Salisbury who "calculated that the number of possible amino-acid sequences is **greater by many orders of magnitude** than the number of proteins which could have existed on Earth since the origin of life, and hence that functionally effective proteins have a vanishingly small chance of arising by mutation". The quote is given from (Maynard Smith 1970) who goes on to illustrate his argument that "evolution has occurred either by the natural

selection of favorable mutations or by chance fixation by genetic drift of selectively neutral mutations" through the following very interesting word game. The purpose is to convert the word WORD to the word GENE through the minimum number of steps **using intermediate words that are meaningful** in English. And the steps are:

WORD WORE GORE GONE GENE

Maynard Smith compares the words to proteins and the letters to amino-acids and the requirement for meaning corresponding to the requirement for functional proteins as intermediary steps. But proteins are coded by genes and for a new protein to arise a gene has to be duplicated and afterwards mutated through slightly favorable or neutral mutations until a new very useful protein is discovered. In other words, "if evolution by natural selection is to occur, functional proteins must form a **continuous network** which can be traversed by unit mutational steps **without passing through nonfunctional intermediates**".

The concept of Computational Habitats has occurred prior to learning about Maynard Smith's concept of protein space but the similarity with our own ideas for searching the solution space is quite striking and I felt all these quotes would help understanding what we want to achieve. Computational Habitats try to be environments where **networks of solutions** (of increased complexity) emerge and evolve through natural selection to match **networks of problems** of equivalent complexity. The reason we use the term "networks" is to emphasize the signaling capability of these populations of problems and solutions.

Although the task is daunting and many questions can be immediately raised (which we'll do and try to answer in the next section) we feel encouraged by Holland who did not originally envisage GAs as functional optimizers, but rather as processes similar to natural adaptive systems (Holland 1962,1992). In his 1962 paper, historically considered to be carrying the primordial ideas behind the Genetic Algorithms concept, Holland defines a "rich" environment as follows:

"In the simplest sense an **environment is rich** when it is **composed of graded sequences of problems**. By its very nature, such an environment permits the adaptive system to develop subroutines for problems at one level of difficulty which, through recombination, minor changes, insertion, etc., will be useful in solving problems at the next level of difficulty". And Holland continues later in the same paper: " an adaptive system could enhance its rate of adaptation by somehow **enriching the environment**. Such enrichment occurs if the adaptive system can generate subproblems or subgoals whose solution will contribute to the solution of the given problems of the environment." Holland's paper, written 43 years ago is the closest model to the concept of Computational Habitat. If geometrically, evolution follows the shape of a spiral, perhaps it is just fitting that Holland started with natural adaptive systems in "rich" environments and ended up defining Genetic Algorithms while we started with GAs and after some stops along the way ended up defining Computational Habitats.

Going back to the issue of search spaces, there is a second step nature has undertaken in reducing the search space and that is the use of genotype to encode a certain phenotypic structure. The widely (still) used method of using genomes as phenotypic structures or of deploying a fixed genotype to phenotype mapping restricts evolutionary algorithms from being used in problems of ever increasing complexity. Rather than using one method for simple problems and another method for complex problems the CH agents will always have a genotype and a phenotype.

Other known issues related to GAs are for example genotype encoding. Fixed genotypes are specifically problematic. With such an approach one wouldn't even think of finding a path in solution spaces of lower dimension (which more likely would involve shorter genotypes). In addition to that, the genotype to phenotype development process has been understudied.

How about the decision of what genetic operators should be used and what should be the values of the probabilities associated with them? Wouldn't it perhaps be better to let the model try various operators and let the more effective ones (for the problem at hand) be discovered?

The choice of selection methods and diversity issues are also taken care of in an autonomous multi-agent model based on natural selection

Table 1: A summary of known issues with classical evolutionary algorithms.

<i>Issue</i>	<i>Description</i>
Solution complexity	The search of high complexity problem spaces starts with a random population of high complexity solutions. Solution space often too large.
Genotype-phenotype	Very often no phenotype. Same issue with searching a very large search space without benefiting from genotype compression.
Genotype encoding	Fixed size in many instances. Almost no research into importing development techniques derived from nature.
Genetic operators	Not always clear which ones should be used for a certain application. Almost always experimentation with operators' probabilities is required.
Selection methods	Many selection methods proposed but none works well in all cases. Hard to achieve a balance between maintaining selection pressure and diversity.
Speciation	Lack of speciation. Populations tend to converge to unique species.
Cooperation	Requires special modeling (e.g. cooperative coevolution).
Learning	Learning (phenotypic learning that is) has been studied but it is not a component of usual evolutionary algorithms.
Hierarchical solutions	Possible only through certain special GP techniques.

1.3 The Contributions Made by this Thesis

1.3.1 Competitive-Cooperative Coevolution Genetic Algorithm

Searching a high dimensional search space starting with a population of randomly initialized (complex) solutions becomes an insurmountable problem once the complexity of the applications goes above a certain threshold. The problem has been recognized by the research community and work for dealing with complexity was started at the beginning of the 1990s. Some of the solutions include **incremental evolution** (equivalent monikers being behavioral memory, sequential evolution or layered learning) (de Garis 1990, Harvey 1992, 1994, Schoenauer & Xanthakis 1993 and others) and **cooperative coevolution** (Potter & De Jong 1994, 2000).

Our own research in this area has resulted in a significant contribution to cooperative coevolution. The introduction of new species is a trial-and-error process in the classical cooperative coevolution model. The Competitive-Cooperative Coevolution Genetic Algorithm (CCC-GA) that we introduced in (Tulai & Oppacher 2002a) deals with this aspect by replacing that ill defined phase of introducing new species with a competitive phase based on parallel GAs. In Chapter 3 we give a detailed description of this algorithm.

1.3.2 Parameter Free Algorithms

An important criterion when choosing an algorithm is its ease of use. Algorithms that require fewer parameters to set are often preferred, especially by new users. In practical problems, like data classification, users want to reliably find out what the accuracy of a classification algorithm is without having to spend weeks in learning on how to manipulate various algorithm parameters.

While we believe that ultimately Computational Habitats will provide a general solution to this very important issue, our interest and research in this area has materialized in two separate contributions. First we have developed an algorithm that treats the strategy parameters (mutation and crossover probabilities, etc.) as genes and evolves them together with the problem specific parameters. We have successfully used this algorithm in problem optimization (Tulai & Oppacher 2002b). Second, and more importantly, our own data classification algorithm called Multiple-Species Weighted Voting (MSWV) is a relatively parameter-free algorithm (Tulai & Oppacher 2004c, Tulai & Oppacher 2004d) in the sense that all the experiments have been performed using the same parameters. In Chapter 3 we give a detailed description of this contribution.

1.3.3 Manual and Automatic Speciation

As mentioned in Table 1 one of the problems associated with classical Evolutionary Algorithms is the lack of speciation. Populations tend to converge to unique species and long-term loss of diversity is a real problem. While the CH model allows for new species to emerge, evolve and coexist naturally, our interest and research in speciation has predated the idea of

Computational Habitats. In (Tulai & Oppacher 2002a) the speciation was performed manually with new species carrying information describing the neurons used in growing the cascade neural network (like neuron characteristics, weights on input connections, etc.).

In (Tulai & Oppacher 2004a) the speciation was also performed manually with individual species of decision trees being assigned to each class in the dataset. This model to which we were led by our interest in speciation proved to be very insightful as we subsequently discovered that that is the standard method used by Support Vector Machines (SVM) for dealing with multi-class classification problems. (Vapnik 1998). Moreover, this particular model has subsequently triggered and facilitated a chain of very significant improvements making MSWV the best GBML in the field (based on the results given in section 3.3.3.). One of these improvements is an innovative automatic speciation mechanism that was introduced to preserve the diversity of decision rules and accelerate their discovery (Tulai & Oppacher 2004c, Tulai & Oppacher 2004d).

Chapter 3 contains a detailed description of the MSWV system. Using thirty-six real-world learning tasks we show that MSWV significantly outperforms (over 97% confidence) a number of well known classification algorithms (NB, IB1, C4.5, PART) and is outperformed by SMO, an SVM based algorithm, with 55% confidence. When the comparison is conducted on a subset of 17 datasets containing at least one continuous numerical feature MSWV significantly outperforms all the classification algorithms it was compared to. MSWV is between 3 and 4 orders of magnitude slower than the other algorithms when the comparison is done on datasets with less than 1000 instances but it scales as $O(N)$ while SMO scales as $O(N^2)$ (see the discussion on speed and scalability in section 3.3.5).

1.3.4 Computational Habitats Theoretical Framework

The Computational Habitat (*CH*) model has its roots in π -calculus (Milner et al. 1989), the Chemical Abstract Machine (Berry & Boudol 1992) and the generative communication methodology of Linda language (Gelernter 1985).

The main characteristic of the *CH* model is the **stochastic nature** of its agents. When these agents have no computational component they are called **stochastic agents** and the habitat will be simply called **Stochastic Habitat (*SH*)**. The second fundamental characteristic of the *CH* agents is **signaling**.

If the **atom** has been considered for a long time to be the smallest particle of matter, a **stochastic process with signaling ability** is the smallest particle of the *CH* model.

Because the stochastic agent's phenotype is composed of many such stochastic processes running in parallel, signaling characterizes the functionality of the agent internally and externally.

In the previous section we mentioned that agents in *CH*s will have a genotype and phenotype. A third major characteristic of the stochastic agents is the use of an **artificial embryogenesis** (development) process for generating the phenotype from the genotype. As it will be argued in section 3.4 the stochastic and signaling characteristics are strong premises for implementing complex genotype to phenotype development processes.

Although the *CH* mathematical framework is an ongoing process the elements defined in Chapter 4 have provided a solid foundation for the first habitat implementation.

1.3.5 First Stochastic Habitat Implementation

Following the research lines described in our Thesis Proposal (Tulai 2003) the first implementations of a *SH* are described in Chapter 5 and 6 together with the studies of evolution and emergence made possible by these implementations. The two chapters contain many important results amongst which we would like to mention the following:

- The life expectancy of populations of stochastic agents closely follows the Gompertz theoretical law for human populations
- The stochastic agents' phenotype is the result of a detailed development process

- Emergence of signaling path differentiation through evolution
- The abovementioned development process has facilitated the natural implementation of an organism repair mechanism in stochastic agents

1.3.6 Data Classification in Computational Habitats

Paralleling and providing an insight into the design of MSWV system, the first results within the new *CH* framework are given in Chapter 7 where we present CHC (CH-based classifier).

2 STATE-OF-THE-ART LITERATURE

"And as he thus spoke for himself, Festus said with a loud voice, Paul, thou art beside thyself; much learning doth make thee mad." - Bible

A search of the AI literature has failed at revealing a previous use of the term "Computational Habitats". However, motivated by the increased connectivity of the computer networks, by the distributed computational systems and by their growth towards becoming a **community of concurrent processes** the term "Computational Ecologies" has been used before in association with such models (Huberman & Hogg 1992). The goal of that research was to improve the overall system performance by optimizing resource contention and the reward mechanisms (through external control). In spite of name and meaning similarity the authors describe a different model with different objectives.

In this chapter we review the state-of-the-art literature relevant to the Computational Habitat concept and such a review unavoidably starts at von Neumann.

2.1 Turing and von Neumann

A brilliant mathematician, prominent figure in the field of Computer Engineering and pioneer of Artificial Intelligence, von Neumann spent his last years of life working on the theory of automata. With the ambitious goal of developing a universal constructor, a machine capable of building any machine given its description, von Neumann was creating the premises for self-replication. Von Neumann was the first to use the concepts introduced by Alan Turing to design a cellular self-reproducing automaton. His design (von Neumann 1966) contains a Universal Turing machine and a cellular array. Each cell is a 29-state machine and uses the 5-neighborhood (the von Neumann neighborhood). His cellular machine constructs any machine described on its input tape, constructs a copy of the input tape and also attaches the two, thus finishing the replicating process. Self-reproduction occurs when the machine described on the tape is the universal constructor itself.

While universal construction capability may not be a requirement for self-replication as Langton pointed out (Langton 1984), it may very well be another prerequisite for AI, not in the sense of a test like the Turing Test (Turing 1950), but in the sense of potentiality. An individual does not possess universal construction capability; however we may say that humans have universal construction capability. What's important is the potentiality of evolving this capability **given the right conditions**. While self-replication is very important, a system must possess **unbounded evolutionary capability** (Bedau et. al. 1998) for hope to exist that it carries the seed of AI and universal construction.

2.2 Deterministic Cellular Automata

Deterministic Cellular Automata (CA) and restrictions of them (like Partitioned Cellular Automata – PCAs) have been used over the last 20 years to build self-replicating patterns (like loops, worms, etc.). As we shall see, these self-replicating (SR) patterns have been also shown to be able to carry computations.

Langton's SR loop (Langton 1984, 1986) was the result of a work meant to show that self-replication can be achieved with considerably less complexity than von Neumann's machine. Interestingly enough the loop he designed had a genotype and also a phenotype. In Langton's SR loop the translation of the signals traveling through the loop (the "genome") is performed when the instruction signals are executed as they reach the end of the construction arm. The transcription is achieved by duplicating the signals at the arm junction. In his paper Langton correctly points out (as we have previously mentioned) that von Neumann's condition "that any self-reproducing configuration must be capable of universal construction" is not necessary because "it eliminates all naturally occurring self-reproducing systems as well". Langton makes these comments before introducing his self-reproducing artificial system, the loop. While the statement is correct, it suggests placing the self-reproducing loops on the same level with the naturally occurring self-reproducing systems. This suggestion is misleading. The major difference between the two is that while earliest SR molecules proved (that given the right environment) they can evolve towards an organism with universal construction capability (if we accept that humans do possess this capability), not the same thing can be said about

Langton's loop. The difference is less in the organisms themselves or the fact that one has naturally emerged and the other has been created by the human mind and more in the environments in which they live and replicate. Even if we remove the boundaries on the two-dimensional cellular matrix and the computational power required to keep the cells replicating it is quite certain the outcome is going to be nothing but clusters of loops and broken loops. This is a major reason why we have rejected deterministic Cellular Automata as the environment to model the Computation Habitat concepts.

Langton's loop has triggered an intense research in Artificial Life. Researchers in the field have first tried to go one step further and simplify his loop. Through elimination of the inner wall of the loop, a significant reduction from 8 states per cell, 72 cells and 108 transition rules to 6 states, 12 cells and 57 rules was achieved (Byl 1989).

A big step towards an early form of Computational Habitat (although not explicitly called so) was taken when rudimentary computational capabilities were added to the SR loops (Tempesti 1995). Tempesti added computational capabilities to a modified version of Langton's SR loop. Among the major modifications made to the loop we can count:

- the use of a 9-neighborhood instead of a 5-neighborhood
- the elimination of the external sheath and enlargement of the inner space
- the construction of 4-different children done in parallel rather than serial
- by simplifying the genome some signaling cells were freed for carrying a program
- due to a protection mechanism the loop did not crash when the construction arm found another loop in its way or reached the end of the cellular space.

In his paper Tempesti gives an example where the functional instructions added to the genome simply display three letters inside the empty space of the loop (after a special instruction

creates a breach/door in the inner sheath). Tempesti's stated goal was to build a Turing machine but that goal was not achieved until a year later (Perrier et. al. 1996) thus creating a SR universal constructor. The program and the data tapes were attached to the main loop and could grow downward indefinitely (as required by a Turing machine). Instead of Turing's transition states Perrier et. al. used Wang's instruction set. Each cell could be in one of 63 states making for a staggering 63^5 entries transition function, out of which only 8503 entries lead to a state change. The paper did not bring any novel idea; it just cleverly put together already existing concepts to come up with something easier to understand than a von Neumann machine but just as unusable practically.

All the SR patterns discussed above (so until around 1996) were not emerging and they were not moving. There was also no guarantee these SR patterns would disappear from the environment. For example in all previous reports the simulations ended with a cellular space full of dead SR patterns (Figure 2 shows the initial and the final Langton SR loops). The initial environment had to be seeded with one SR loop and was no guarantee it could emerge. To understand why, we give this quote from Stephen Wolfram (Wolfram 2002). "One consequence of local irreversibility is that some cellular automata configurations may appear as initial conditions but may never be reached as descendants of other configurations through cellular automata time evolution."

Figure 2: Initial and final (dead) Langton self-replicating loop

Intuitively by solving the irreversibility issue SR patterns could be made to move. By working in a framework of Partitioned Cellular Automata (PCA), a subset of CAs, this was shown to be

indeed feasible (Morita & Imai 1996). In PCAs each cell is divided in 5 partitions (sub-cells). Each partition (in their case) can have 8 states so a cell can be in one of 8^5 states. The most important property for PCAs is that injectivity of the global map is equivalent to injectivity of the local map. Because of this property the PCA framework can be used to design Reversible Cellular Automata (RCA) which is an injective CA (any state is guaranteed to have at most one ancestor only). Consequently the task of designing an RCA is reduced to the task of designing an injective local PCA rule map. The reversibility property makes it easy to design worms and loops that move and self-replicate. This is achieved through self-inspection of the tail cell and generation of commands for extending the structure through its head. Self-replication is achieved through branching (instead of one new cell, two or three new cells are generated at the head of the worm). If reversibility is abandoned (while preserving the PCA environment) the rules for the self-replicating moving worms can be considerably simplified (Morita & Imai 1997) showing that reversibility is just a framework for facilitating the design but it is not essential for such structures to exist. Self-replication through self-inspection has also been used by other researchers in the field (Ibanez et. al. 1995) but their work was poorly explained (rules are not explained, step by step examples not given etc). For a historical perspective we have to mention that self-replication through self-inspection was first used in the context of Turing machines (Laing 1977). By enhancing the Wang version of the Turing set with a special instruction Laing came up with a set of special Turing machines that can be used to show how a machine can self-replicate itself.

In (Chou & Reggia 1997) for the first time was shown that SR structures can spontaneously emerge from a primordial soup. Their SR loops could also grow in size due to a GROWTH bit left behind by one of the genomic bits of a dying loop. The GROWTH bit was actually tied to a certain location in the environment (where the previous death had occurred) and might be later "consumed" by another SR loop hovering over that location. After the GROWTH bit was consumed (leading to an extended loop) it was removed from the environment. Their work is very interesting (because is providing for the creation and dissolving of loops, for loop growth – although no loop shrinking – phenotype and genotype bits) but the emergence of

2x2 loops (the first loops to emerge) requires a special set of rules to detect the presence of the necessary blocks of the loop and set the BOUND bit.

Just as Tempesti has added some computational capability to non-emerging SR loops, problem solving capabilities have been added to SR loops (Chou & Reggia 1998). Their problem solving SR loops represented an early form of Computational Habitat. Chou and his collaborators have used SR loops and their habitat to solve the SAT-problem. Each SR loop carried the exact number of bits required by a given predicate equation (m bits for an m -SAT problem) and various clauses (of the generated predicate) were embedded into the cellular structure destroying the loops sitting on the top of them when they failed to satisfy that particular clause. The bits of the loop that were not used as signals during the self-replication process could be used to carry bit sequences representing Boolean assignments for a SAT problem. The replication process was designed such that each child would have one explored bit different from its parent (initially there was one loop in the cellular space carrying an unexplored sequence of bits, for example x 's). In the end there would be one or more loops in the cellular automata space carrying a different fully explored bit sequence. In between the start and the end there was an enumeration process during which the various explored bit sequences were selected (and possibly removed) by monitoring mechanisms built into the cellular space. The combination of enumeration and selection mechanisms was similar to the one employed by DNA computing and suffered from similar problems (for example regions of the cellular space may get crowded with unchecked loops preventing further explorations).

As previously shown in Figure 2, in Langton's experiments the cellular space filled up with dead empty loops preventing further evolution of the system. In nature dead organisms don't simply lay around unchecked, they decay. Through decay resources owned by dead organisms will be re-utilized by other systems. This provides the whole system with a negative feedback loop and increased probability of evolution. In the case of Langton's SR loops the resource that was required for continuing the evolution was the cellular space. By introducing an additional state into the set of states of the CA, a structurally dissolvable self-reproducing (SDSR) loop has been created (Sayama 98). When faced with critical situations, like shortage of

space for self-reproduction or collisions, certain cells within the loop would enter the dissolving state and propagate through the rest of the loop dissolving it. The space created by dissolving loops was used for continuing the evolution of the system. Sayama also showed that loops of a certain size had better chances of survival when competing with loops of a different size. Studying the collision of the loops leading to the formation of new larger loops Sayama made an interesting observation that should be carefully considered in emergence studies, I quote: "...at the ancient dawn the genotypes of living objects may have been physical structures at the same scale as their phenotypes, so that the genotype may have had some phenotypical character, and vice versa. Thus, it is very likely that evolution of life at that time was accomplished not only by change of genotypes but also by such environmental factors as direct interaction of phenotypes---in other words, such environmental factors could alter how the genotype is interpreted into a phenotype". In other words, Sayama suggested, at a certain scale environmental factors may act as rybozymes. The same idea was expended upon in subsequent work (Sayama 1999).

To conclude we are going to review the definition of a two-dimensional CA with 5-cell neighborhood (following Morita & Imai 96) and make an observation.

Definition: A deterministic two-dimensional cellular automaton with von Neumann neighborhood is a system defined by $A = (Z^2, Q, f, \#)$ where Z is the set of all integers, Q is a non-empty finite set of internal states of each cell, $f : Q^5 \rightarrow Q$ is a mapping called a local function, and $\# \in Q$ is a quiescent state that satisfies $f(\#, \#, \#, \#, \#) = \#$. A configuration over Q is a mapping $\alpha : Z^2 \rightarrow Q$. Let $Conf(Q) = \{\alpha \mid \alpha : Z^2 \rightarrow Q\}$ denote the set of all configurations over Q . The global function of A , $F : Conf(Q) \rightarrow Conf(Q)$ is defined : $F(\alpha)(x, y) = f(\alpha(x, y), \alpha(x, y + 1), \alpha(x + 1, y), \alpha(x, y - 1), \alpha(x - 1, y))$, $\forall (x, y) \in Z^2$.

Let's now make the notations:

$$\alpha_u \stackrel{def}{=} \alpha(x, y + 1), \alpha_r \stackrel{def}{=} \alpha(x + 1, y), \alpha_d \stackrel{def}{=} \alpha(x, y - 1), \alpha_l \stackrel{def}{=} \alpha(x - 1, y), \alpha_c \stackrel{def}{=} \alpha(x, y)$$

We now consider a Computational Habitat with a cellular array landscape with each cell populated with one agent with four input and four output ports. Each agent has only one sub-agent, a computational one. The agent placed at location (x, y) is represented by the equation:

$$\left(\frac{u_{x,y+1}(\alpha_u) \cdot r_{x+1,y}(\alpha_r) \cdot d_{x,y-1}(\alpha_d) \cdot l_{x-1,y}(\alpha_l) \cdot \bar{d}_{x,y} \alpha_c \cdot \bar{l}_{x,y} \alpha_c \cdot \bar{u}_{x,y} \alpha_c \cdot \bar{r}_{x,y} \alpha_c \cdot f(\alpha_c, \alpha_u, \alpha_r, \alpha_d, \alpha_l)}{1.0} \right)$$

In other words the computational sub-agent implements the local state machine of the CA and all the agents are identical up to four input and output ports which are specifically labeled to suggest the communication is restricted to the von Neumann neighborhood.

So the observation we now make is that **deterministic Cellular Automata are special cases of Computational Habitats** as they will be defined in Chapter 4.

2.3 Non-Uniform Cellular Automata

One would have expected (as I have) that research into the self-replication and computational capabilities of **non-deterministic** (i.e probabilistic) CAs must have naturally complemented the large amount of research in deterministic CAs. If it's out there it has eluded me. However there is some research and results using non-uniform cellular automata (NUCAs) and most of it has been done by Moshe Sipper. I'll review some of his papers that pretty much summarize the progress in this small (but important) field.

NUCAs share the cellular landscape with deterministic CAs but their local rules are not uniform (see definition of deterministic CAs at the end of previous section). In fact a mathematical definition of the NUCA used in (Sipper 94) is very difficult if not impossible. There may be cells with no rules and cells with one or even two rules (one active and one inactive). The rules could change (based on a 9-neighborhood) and also could move (!) to empty cells (in the neighborhood). The rules of one cell could also alter the rules of a neighborhood cell (Sipper 94). Using this eclectic methodology the following results were reported:

- self-replication of loops **seeded** in the cellular space

- copying of passive structures by active (mobile) cells
- starting with a primordial soup of builders and replicators, formation and replication of complex structures was reported

If we take one step back and we consider just the case of one-rule per cell with von Neumann neighborhood the definition of NUCAs can be obtained from that of deterministic CAs by replacing the universal local rule f with a family of rules indexed by the location (x, y) . We get: $NUCA = (Z^2, Q, f_{(x,y)}, \#)$ with $f_{(x,y)} : Q^5 \rightarrow Q$.

From the above it becomes clear thus defined NUCAs are special cases of Computational Habitats but are generalizations of CAs.

Because, starting with von Neumann, it has been proven in various contexts (different number of states per cell and neighborhoods) that two and one-dimensional CAs could support universal computation it was expected and proven that NUCAs, as generalization of CAs, also had this capability (Sipper 1995). Aside from proving the universal computational capability of NUCAs the paper also proved the more interesting universality of a 2-state, 5-neighbor NUCA (Codd had empirically shown that 2-state, 5-neighbor CAs did not have the universal computation capability for finite initial configurations).

As generalizations of NUCAs and CAs, Computational Habitats on 1 or 2-dimensional cellular landscapes can also support universal computation.

A well known computational task deterministic 1-dimensional CAs have been used on is the $\rho_c = 0.5$ density task (Mitchell et al. 1994). In that research the rules of the CA have been designed using a GA. 1-dimensional NUCAs have been used on the same task but the cell rules rather than globally designed were locally evolved (Sipper 1996). While the best results for $k = 2$ and $r = 3$ (2 states per cell and radius of 3 neighborhood) reported using NUCAs are slightly worst than those reported using the uniform (deterministic) CAs with some runs showing a performance close to that of a random run (i.e. no improvement through evolution)

the interesting finding is that research on the same task using $r = 2$ and $r = 1$ neighborhoods have shown surprisingly good results when compared (for example in the case of $r = 1$) with the best results a CA could possibly achieve. The NUCA achieved these results not with a single rule but **with a combination of rules** which shows that in certain problems improved solutions can be achieved through collaborations.

The communication capability of Computational Habitats clearly extends beyond some local connections. Would this capability help with the density task? Using NUCAs and evolving the architecture of the cell connections it was shown that previous positive results on the density task have been further improved (Sipper & Ruppel 1996). It is unfortunate this paper is weak statistically, limiting itself to displaying the result "of a typical run". Nonetheless the results further confirm the potential of the Computational Habitats.

To conclude this section we would like to point out that unlike CAs and Computational Habitats (as intended), some of the research conducted in NUCAs has used external control (for example keeping fitness scores per cell, determining whether crossover is applicable or not, selection of candidates for mating, etc.)

2.4 Artificial Chemistries

Although the emergence of Computational Habitats is more concerned with the emergence of the computational sub-agent component, the refinement of the model would benefit from further developments into the emergence of self-replication and (artificial) life in general. Artificial Chemistries (ACs) could provide just the right environment and tools for such research to be conducted drawing a parallel with the role of real chemistry in the emergence of life.

The research in this area has been quite intense not only for its potential of shedding light on the dynamics of early life but also for its application potential for modeling, information processing and optimization (problem solving), the last one being closely related to the field of evolutionary computing (Dittrich et al. 2001) and (in our view) to Computational Habitats. An AC model is defined by (1) a set of molecules (e.g. binary strings, abstract symbols, numbers,

proofs, finite automata, lambda expressions, Turing machines, etc.) (2) a set of rules describing the interaction between molecules and (3) a domain and algorithm determining how the rules are applied to the collection of molecules.

The link between artificial chemistries and concurrent processes theory has been captured in the Chemical Abstract Machine (CHAM) concept (Berry & Boudol 1991) which is closely related to the CH language as shown in Chapter 4. Another strong link is provided by the lambda-calculus (Barendregt & Barendsen 1991, Barendregt 1992) which was proven to be embedded by both π -calculus and by the CHAM and it was used (lambda expressions that is) as "molecules" mixed in a "Turing gas" as a primordial "soup" for emergence studies (Fontana 1991, Fontana 1992, Fontana & Buss 1994). These two links would suggest that further research into emergence of the "natural" sub-agents components of the stochastic agent (the *CH* agent), could be performed in an environment rich in "inert" but reachable (see the note on self channel in section 4.1) lambda expressions. This may provide a very interesting line of research in the future.

In the studies performed by Fontana, the initial population of lambda expressions was evolved following some precise, but arguable rules. For example the result of the collision between *A* and *B* is not the same with that between *B* and *A*, so it is not commutative. The catalytic reactions between *A* and *B* not only produces a compound object $A \odot_{\Phi} B \equiv ((\Phi)A)B$ (where Φ is a generalized collision rule that may be different from the regular application rule $\lambda x_1. \lambda x_2. (x_1)x_2$) but it also returns *A* and *B* back to the mix (equations are not mass conserving), all interactions taking place in a flow reactor in a stoichiometric framework. In these conditions three types of network emerged (Fontana & Buss 1994):

- **level 0:** the system was initialized with many randomly generated functions and the axioms of the λ -calculus were applied. Elementary reactions at this level included single-step replications (controversial). Because replicating objects could cross-catalyse each other's replication, hypercycles also could arise, but they tended to collapse to simple replicative autocatalysts upon perturbations (introduction of random objects).

- **level 1:** If direct replication was not allowed (external rules), self-maintaining organizations emerged. Despite the fact that there was a continuous outflow, these organizations (specified by a list of objects and the reactions among them) did not vanish.
- **level 2:** they arose from two different level 1 networks. When neither of the networks dominated, a self-maintaining organization emerged that was not a trivial 'sum' of the two component networks: it also contained a connective metabolic part ('glue') that manifested the cross-reactions linking the two (level 1) subsystems

While the results are interesting, the research would have benefited from a stronger familiarity with previous work in biology and biochemistry. Some recommendations for placing their experiments on a better chemical footing are (I quote from Szathmáry 1995b):

- 1) a proper implementation of elementary steps in which no one-step catalyses are allowed
- 2) the allowance for the possibility of multiple instances of the same molecular type at the beginning

Self-replication and self-organization was also proven using PROTEAN, an autopoietic ecology initialized with a population of Turing machines (rather than a Turing gas) sharing a memory space (Yoshii et al. 1998a, 1998b). Their analysis was conducted based on concepts of space (length of programs) and time (time to execute) complexity. The main result of the paper was the fact that given enough time these populations of computer programs acting upon one each other's tape arrived at a sort of balanced equilibrium, although it is not clear if the initial configuration (of 5 machines) was randomly generated or not.

More recently self-replication has also been shown using "squirm3", a system that uses single components called atoms (Hutton 2002). The author considers the "primordial soup" of artificial atoms and their "reactions" rules as an AC. Because each atom has a *type* from a set

{a,b,c,d,e,f} and a *state* {0,1,2, } we can equivalently see the artificial atoms as labeled finite state machines that change their state when they come in the vicinity of other state machines (atoms). Hutton's equations seem to be mass conserving (unlike Fontana's). There are also **bonds**, which are physical connections between atoms. Bonds could be formed and broken by reactions. Two or more atoms connected by bonds form a **molecule**. The environment is a two dimensional grid (just like a CA structure) with each cell being either empty (in other words does not contain an atom) or occupied by an atom (that's good because some physics goes into it as well, unlike the CAs). The atoms move around at random—at each simulation time step each atom either moves to an empty neighboring square or stays still. A move to a square is valid if the atom remains within the neighborhood of every atom it is bonded with.

The self-replicating organisms in this experiment were strings. One interesting experiment also discussed in the paper is the regular (at interval T) "cleansing" of the environment of debris (unused atoms) followed by the insertion of new fresh material (flooding). This experiment was shown to favor the formation of short self-replicating molecules.

Driven by research in nanotechnology an even more recent paper reports replication by assembly and, given enough time, the forming of self-replicating patterns in a primordial soup (Smith & et al. 2003). The particles used in that research were also discrete states automata, but instead of being discreetly placed on a two-dimensional grid they were allowed to flow freely (like in a virtual liquid) and their external relationships were governed by a simulated physics that included Brownian motion, viscosity, and spring-like attractive and repulsive forces. The particles can be assembled into patterns that can encode arbitrary strings of bits.

What we have reviewed so far would (wrongly) suggest that ACs have been used in emergence studies only. ACs offer an environment where multiple calculations (as reactions between molecules) could be performed in parallel, very similar to the CAs and more so to NUCAs. We say more so because just like in the case of NUCAs we can divide the environment in as fine a grid as we need to have one molecule sitting in each cell on its own (with some cells having no molecules) and interacting locally, or remotely (we've seen this considered in a NUCA

framework) with other molecules according to a predetermined set of rules (the rules of the AC). And just like in CAs and NUCAs, ACs can be used to do **information processing**. Starting with a primordial soup (more complex than the Turing gas) containing objects like character sequences, lambda expressions, binary strings or numbers and using a set of reactions or collision rules and an algorithm to run the system, solutions to computational problems like parity checking, sorting algorithms or even prime number generators have been shown to emerge (Banzhaf et al. 1996). Their work was also motivated by the relevancy of this research to practical fields like nanotechnology and parallel computers based on molecular devices.

But the use of ACs as a model for performing computations has also been reflected over the last 5 years in the emergence of a new **natural computing** field called **membrane computing**. The five domains considered to belong to the field of natural computing are: neural networks, genetic algorithms, DNA computing, membrane computing and quantum computing. The first two are established fields and are considered to provide good/approximate solutions while DNA computing and membrane computing are considered to be newer fields that are "aimed at computing in parallel optimal/exact solutions" (Calude & Păun 2001). If the quote is intended to suggest old domains offer approximate solutions while new domains offer exact solutions it would mislead the reader. Quantum computing is also a very new field and it is probably the most promising computational tool. However who is familiar with the field knows the quantum computing solutions have a probabilistic nature (even if the probability of getting the right result can be made arbitrary large it is never equal to 1). From this perspective Computational Habitats clearly belongs to the field of natural computing as a model that seeks approximate solutions in general but may also deliver exact solutions for certain classes of problems.

In the membrane computing paradigm there is a formal representation of the AC as a computing device, that the originator of the field calls P-systems (Păun 2000). The P-system, very similar to an AC, has molecules, rules, a domain similar to a cell structure separated by membranes and an algorithm specifying how the rules are applied. The P-systems are closely

related to CHAMs (Berry & Boudo 1992), Gamma language and the chemical reaction model (Banâtre & Le Métayer 1996) and the reflexive CHAM (Fournet & Gonthier 1996, Fournet et al. 1996). P-systems have been very successfully used for proving many theoretical results. For example characterizations of recursively enumerable languages have been obtained by very simple P systems.

2.5 Computer Programs Worlds

In all the previous sections of this chapter we have reviewed different specific forms of Artificial Life. However there are forms of Artificial Life that do not fall in any of the previous categories but nevertheless are very important as they can also be seen as (early) forms of Computational Habitats. The forms of Artificial Life we are talking about are computer programs.

Leaving the world of CAs but using some sort of computational chemistry (the space was seeded with assembly-like instructions) and a circular memory, a virtual world was created and named **assembler automaton** and the program called "Coreworld" (Rasmussen et al. 1990, 1992). This world was known to have universal computational capability and the goal of that research was to find fixed points of the Coreworld dynamics (i.e. self-replicating computer programs or stable patterns) in the presence of noise (noise was created by a flawed assembly instruction, i.e. an instruction that not always was doing what it was supposed to do, with a certain probability). Although some interesting patterns in the dynamics of the world were found, the Coreworld was considered to be too fragile under mutation (the patterns were very much dependent on random events occurring early in the history of the core).

The best known virtual world in which agents are self-replicating computer programs is Tierra (Ray 1992). Because some (percentage) of the failure of the Coreworld had been attributed to the specifics of the assembly language it used (a language called Redcode previously used in a computer war game), Ray, who was aware of Rasmussen's work, decided to use his own assembly language set with 32 instructions coded in 5 bits. In Tierra the programs competed for execution time and evolved in a simulated parallel environment. The evolutionary aspect of

their behaviors extended to their reproductive mechanisms but they were relying on externally specified mutation mechanisms. Also, the Tierra environment must be seeded with hand-coded replicators. Because the agents in Tierra are computer programs it is clear the parallel with the agents in Computational Habitats can be made. However, the stochastic aspect specific to the CH agents is not present in the Tierra agents. Another significant difference is that is not a problem-solving world. The model is useful for the study of evolution but the Tierra agents do not carry a computational component like the CH agents do.

In the Avida model, the computational capability, missing in the initial Tierra system, was added in the form of an external controller imposing a (very simple) problem on the creatures living in a 2D environment (Adami & Brown 1994, Adami 1995). The external manager supervises and rewards (with additional time to run) the agents based on their proficiency at solving the problem (in their case just adding two numbers). The creatures in Avida do communicate with each others through buffers. This capability of the signaling channels was missing from the original Tierra world and it is interesting to notice that Adami considered it as an important characteristic when adding computational capability to the model. Signaling-facilitated cooperation strategies have also been studied in a habitat environment in the context of resource abundance and high-density populations (Wagner 1999). Internal (within agents) and external (between agents) signaling is a major characteristic of Computational Habitats as discussed in the last section of Chapter 3 and in Chapter 4.

One of the major drawbacks of Avida is that for the strings-of-assembly-language-instructions creatures that live in the cells of the 2D landscape there is no distinction between genotype and phenotype. As we have mentioned in Chapter 1, we see the presence of a small genotype that can develop into a phenotype during a development process as an important method for reducing the size of the search space. That being said, the Avida model is a simplified Computational Habitat with a 2D landscape, without a stochastic component, with genotypes but no phenotypes.

More recently, using Genetic Programming (GP) and a LISP-like language called Push, an autoconstructive evolution system called Pushpop was created. Pushpop programs construct their own children while solving (simple) computational problems (Spector 2001).

One thing that becomes clear analyzing the previous work on CH-like models is that when the creatures spend a lot of time in evolving basic functions (like reproduction) the likelihood of being able to tackle and solve other complex problems is small. This insight suggested that the research in Computational Habitats must start at a point in evolution where these functions (like feeding, reproducing, etc.) are already present but may be modified via a set of evolvable parameters and this was exactly our approach as detailed in Chapter 5 and Chapter 6. One set of such parameters would be for example what percentage of the energy intake would be allocated to the various sub-agents. Equilibrium between these parameters must be reached during evolution because if, for example, not enough energy is allocated to the reproduction sub-agent the agent as a whole will not be able to reproduce itself. To this simplified but flexible model a computational sub-agent may be added **thus focusing the evolution on the computational component** while preserving the interactions with and the impact of the other sub-agents.

2.6 Ants Colonies and Model-Based Algorithms

The role of the landscape in defining a problem or the solution(s) is best exemplified in combinatorial problems, with the most known example being the TSP (Traveling Salesman Problem). **Model-based search** (MBS) algorithms have been very successful in tackling combinatorial problems. Heuristic search algorithms can be divided in two categories, **instance-based** and **model-based** (Zlochin et al. 2001). Most classical algorithms, like genetic algorithms, simulated annealing and iterated local search are the well known examples of instance based search because they build new solution(s) using just the current solution or population of solutions. On the other hand, in model-based search candidate solutions are generated using a parameterized probabilistic model that is updated using the previously seen solutions in such a way that the search will concentrate in the regions containing high quality solutions. In this second category are included the following algorithms: **ant colony**

optimization (ACO) metaheuristic (Dorigo and Di Caro 1999), **stochastic gradient ascent** (SGA)(Meleau & Dorigo 2002), **cross-entropy** (CE) method (Rubinstein 1999), **estimation of distribution algorithms** (EDAs) (Mühlenbein & Paaß 1996, Pelikan et al. 1999).

We have to point out that in the case of Computational Habitats the agents have a probabilistic model while in the case of MBS the parameterized probabilistic model refers to the fact that candidate solutions are constructed using a parameterized distribution over the solution space but they (the candidate solutions that is) do not have a probabilistic makeup. As the model progresses the solutions are modified after the distribution of the solution is updated (based on the results obtained to that point). The four MBS methods are closely related and they employ very similar (in certain cases identical) methods for updating the model parameters. Because of this reason for any parallels we need to draw between MBS methods and CHs, we can safely concentrate our attention on the ACO metaheuristic.

ACO algorithms can be applied to pretty much any discrete optimization problem for which a constructed solution is conceivable and for which the application of classical methods (like dynamic programming) is not effective. These problems would include NP-hard problems, shortest path problems in which the graph's representation may change over time or problems in which the computational architecture is spatially distributed (Dorigo & Di Caro 1999). The application of ACO algorithms to dynamic problems, that is, problems whose characteristics change while being solved, is recent. One such application concerned routing in circuit-switched networks (e.g., classical telephone networks) (Dorigo & Stützle 2001).

Using simple organisms (ants) not capable of finding optimal solutions by themselves, the colony is capable to harness their work such that good quality solutions may emerge through collective interactions. The interaction of the ants is done indirectly via a communication agent called "pheromone". The ants move concurrently by applying stochastic local decisions that use information stored in the nodes they reach (for example what are the feasible options for moving out of that node). When multiple options for moving are available the ants take a probabilistic decision that is influenced by the strength of the pheromone attached with each

possible option. The pheromone acts as a catalyst. A higher concentration of pheromone increases the probability an ant will choose a certain path, and the choice itself will further increase the pheromone concentration as the moving ant will itself leave behind a small quantity of pheromone.

The moving ants can be easily simulated by mobile agents in a Computational Habitat with a landscape tailored by the shortest path problem to solve. The equivalent of the pheromones would be taken by some localized messages attached to the node just left behind. The messages will simply say take "take road X". For the sake of argument let's say a mobile agent (when we say a mobile agent we mean an agent with a mobile sub-component) reaches a node from which the road bifurcates towards further nodes X and Y. When the mobile agent reaches the intersection let's say that m messages were left behind telling it to go towards X (let's say left) and n messages telling it to go towards Y (let's say right). Assuming all the pheromones have identical strengths all the messages will have identical probability to be read. The node information will be a parallel process with m processes communicating "go left" and n processes communicating "go right". This matches well with the practical fact that ACO algorithms initialize all possible paths with identical quantities of pheromones (so the first mobile agent to reach a node will find a set of signals outlaying the possible alternatives). The implementation suggested by the Computational Habitats model would also have another advantage. There would be no need for additional information stored in the node! The processes would describe the alternatives and their probabilities would describe the strength of the pheromones. Pheromones evaporations (used in ACO to avoid a too rapid convergence of the algorithm towards a sub-optimal region) could be implemented by adding an apoptotic component to these signals (see Chapter 6 section 6.5). Again it becomes clear how naturally Computational Habitats can be used to model other systems, in this case the ant colonies.

However, we do not simply propose taking other algorithms and porting them to a *CH* model. A major difference is autonomy. ACO algorithms need external control. Besides pheromone evaporation ACO algorithms also use daemon actions to implement centralized actions that cannot be performed by single ants. One example would be the collection of

global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a non-local perspective. As a practical example, the daemon can observe the path found by each ant in the colony and choose to deposit extra pheromone on the arcs used by the ant that made the shortest path. Pheromone updates performed by the daemon are called **offline pheromone updates** (Dorigo & Di Caro 1999). It is clear that offline pheromone updates are not acceptable in the context of Computational Habitats. However, allowing other forms of communication between agents, or even using other agents, CH may offer a different (and we believe more powerful) environment where solutions to discrete optimization problems can be efficiently found.

2.7 Incremental Evolution

This area of research is of particular interest for our own research into Computational Habitats because of the relation between incremental evolution and problem decomposition.

We mentioned before that a search for optimum in the solution space could be helped by either decomposing the problem into smaller components or by using totally new problems of a smaller complexity but still somehow relevant to the major problem under study. Solutions to the new (hopefully) easier problems could subsequently be used as building blocks or initial points in the search for solutions to the more complex problem. **Incremental evolution** (in certain contexts referred to as **multi-agent layered learning**) exploits the same idea.

One would suspect that incremental evolution, termed by Harvey et al. (1994), was a concept that emerged by accident, when the task of evaluating a population of 30 was taking about 1.5 h per generation! And most likely it was but as we'll see later, the concept was put forward in 1990 (de Garis 1990) when it was termed "sequential evolution" or Behavioral Memory. The genome of each individual in the population was encoding a dynamical recurrent network and a visual sampling morphology (the size and the position of a receptive field within the visual field) the basic components of a control system for a robot. After 15 generations at the end of which the "objective" fitness function could not differentiate between 30 solutions, the authors visually picked up one control function that generated a more interesting behavior and

created a new population of clones from this incomplete solution. In less than 10 generations starting from the population of clones a solution for a general problem (a very big white target the robot had to move towards) was reached. The incremental evolution concept was used again when the population arrived at after 12 generations of training on the big task was used as initial population for training on a much harder task (a much smaller white target). The individuals evolved for 6 generations using this method where compared with those arrived at by starting with the population of clones and training directly on the hard task (in other words bypassing the intermediary step of training on the big target). The individuals trained using incremental evolution outperformed those trained directly on the hard task (Harvey et al. 1994).

Would this method always work? Justifying the use of converged populations (on a different task) the authors made the comment: "One could start with a randomly generated population which would be the normal GA technique. But often in a normal GA problem, different parts of the genotype contribute semi-independently to the evaluation function, and through the Schema Theorem (Holland 1975) progress of some sort can be made from such a random start." In other words in those cases where there is a relatively strong relation between the building blocks recombination of genotypes is not of much help. Starting with an already converged population and using some sort of local climbing (through mutations) may be a better solution in these cases.

There is another point reported by Harvey et al. that is worth mentioning because of its relevance to our work. Due to a SW error one of the tactile sensors attached to the robot, report the authors, was acting as a source of noise. The authors went to comment that the unintended source of noise has been exploited by evolution to produce a tightly self-regulating system that generalized very well (performed very well on moving targets also).

Besides the humoristic aspect of the fact that this finding reinforces my own motto: "The best genetic operators are the SW errors!" there is a more profound (and serious) aspect of this finding. In the *CH* model we do not need to make SW errors because the sources of noise

are embedded straight into the stochastic aspect of the agents. Noise has a positive aspect in building robustness into solutions and many times it is injected into the system for that purpose (for example Neural Networks). In the case of Computational Habitats noise is an inherent feature of the model.

Incremental evolution has been systematically studied on a tracking problem (a moving object must be kept as close as possible to the centre of a camera following it) and the results compared with those obtained training a population from scratch, in a GP framework (Winkeler & Manjunath 1998). The incremental evolution strategy adopted is to train first on one portion of the problem only which the authors called **naïve dimension reduction**. The search is encouraged by using demetic grouping (the entire population is divided into sub-populations training in isolation or with migration turned on). The migration between demes was turned off during the initial increment training and was turned back on during later increments and during the final run. During the final run the populations are combined. Without using incremental evolutions we have also experienced good results with initial populations distributed in demes that are eliminated one by one to create a final combined population (Tulai & Oppacher 2002a, Tulai & Oppacher 2002b). The details of our results are presented in Chapter 3. The results of the studies on the tracking problem show the incremental evolution as being a more efficient method when compared to training from scratch (from a randomly initialized population), while the quality of the solutions was as good or better. In no case studied the quality of the solution obtained through incremental evolution was worse than that obtained through training from scratch. Generally the best results have been obtained when each deme was training for a different dimension of the tracking problem, which the authors call **mixed incremental training**. The use of mixed incremental training is seen as a method for avoiding poor rewards or problem division user decisions (Winkeler & Manjunath 1998).

Layered learning approach (the equivalent of incremental evolution in a multi-agent model) facilitates the discovery of primitive behaviors that can be reused and adapted towards complex objectives based on a shared team goal. The idea dates back to 1990 when it was

termed Behavioral Memory or "sequential evolution" (de Garis 1990) as we have previously mentioned. So the general concept of incremental (or sequential) evolution has been indeed formulated as a result of work involving the use of a GA for evolving weights for a fully connected neural network for the control of a stick legged robot. The task for the controller was to have the robot walk from point X towards a point Y, as far as possible from X in a given number of cycles set by the user. However this was done in 3 phases. In the first phase no constraints were imposed on the stick legs (just move towards right as far as possible). In the second phase the stick legs had to move such that the hip joints were above the floor. In the third experiment, a full set of constraints was imposed to ensure a lifelike walking motion (e.g. hip, knees and toes above the floor, knees and toes below the hip, knee angles less than 180 degrees etc). However the 3 phases had something in common. The population of chromosomes evolved during the first phase was used as initial population for the second phase and the final population of second phase used as initial population for last phase. The excellent results obtained lead to idea of "sequential evolution" or Behavioral Memory as it was formulated by de Garis (de Garis 1990).

But the applicability of incremental evolution has not been limited to complex control problems of the kind one would encounter in robots. In GA applications, the most general method to treat constraints is to include them in the objective function and treat the problem as an unconstrained problem. However, this method works when the landscape is smooth and the feasibility region large. For harder problems one approach proposed was based on de Garis' Behavioral Memory, the other moniker for incremental evolution (Schoenauer & Xanthakis 1993). By starting with an initial random population and incrementally training for each of the constraints we practically narrow the domain to the feasibility region. This population of feasible solutions is finally evolved using the objective fitness. This study is important because it shows that the principle of incremental evolution (or its equivalent, layered learning) can be applied to general optimization problems and that will be another line of research that we intend to follow in the future using Computational Habitats.

While constraints in most GA applications are usually fixed (static) the behaviors which one may see as necessary (or unavoidable) constraints are usually dynamic. For example can one play soccer (well) without learning how to pass? However passing, when learned in teams of two or three static or moving people may not exhaustively cover all the aspects required for playing soccer or keepaway soccer. Layered learning has been used in a static configuration with new behaviors learned on top of old behaviors that were kept fixed (Hsu & Gustafson 2002) but it has also been recently used in a dynamic configuration called **concurrent layered learning** (Whiteson & Stone 2003). Concurrent layered learning borrows from cooperative coevolution an evolutionary computation technique (Potter & De Jong 2000) that we have also used extensively in our previous studies (Tulai & Oppacher 2002a).

To conclude the section we would like to point out that the use of the word "learning" in "layered learning" is a misnomer. In all the papers using the layered learning technique that we quoted the weights are not modified through an on-line learning algorithm (like back propagation for example) but are rather evolved through various evolutionary techniques. Therefore layered learning is nothing else but incremental evolution techniques applied to neural networks weights (just as in the original work of de Garis 1990 or Harvey et al. 1992, 1994). Concurrent layered learning is nothing else but incremental evolution combined with cooperative coevolution, and I would rather call it **incremental coevolution** (because the increments keep coevolving as their number increases).

And as a final comment, incremental coevolution (as defined above) is inherent to a Computational Habitat model where agents evolve autonomously.

2.8 Multi-Agent Systems

As it will be defined in Chapter 4 a Computational Habitat is after all a multi-agent system so a brief review of this area is needed. We say brief because multi-agent systems (MAS) is a very large area indeed and an exhaustive study is not feasible, therefore we'll concentrate on MASs used for problem solving. This application area can in itself be further divided in two sub-areas: distributed solving of problems and solving distributed problems.

The two problem areas mentioned above can also be considered in a Computational Habitat model and what interests us is the approach used. The classical methods are more concerned with the information and knowledge exchange framework to support the distributed problem solving and address, for example, issues like how the agents should be connected (Luo et al. 2001). From this point of view the problem solving technique offered by the Computational Habitat model is closer to a distributed technique for problem solving known as eco-problem solving. Eco-problem solving is a technique that relies on coordination rather than on distributed expertise or distributed domains (the case of physically distributed systems, e.g. problems where the task is to control a communications or energy network). So what are the similarities and differences between CH and ECO (a general model based on eco-problem solving methodology - Ferber 1999)?

Similarities

1. The agents act autonomously.
2. Solutions are arrived at by each agent trying to achieve its personal goal.
3. The plan for solving a problem is obtained as a secondary effect of the behavior of the agents.

Differences

1. In ECO the agents are finite-state automata while in CH they are implemented as concurrent stochastic processes. It turns out the CH agents' model is a generalization of the ECO agent' model.
2. In ECO the agents are fixed, they do not evolve and are not subject to selection. In CH the agents reproduce and evolve and they are subject to natural selection laws.
3. In ECO the agents have fixed behaviors which are pre-determined by the designers during an initial phase. In CH the behavior of the agents evolves.

4. ECO is more appropriate for constraint satisfaction problems. Just like EC in general, CHs (we believe) can be applied to any problem.

It is quite clear that the differences between the two models are fundamental. One interesting characteristic of eco-problem solving systems is their robustness to noise. This characteristic should be also shared by CH systems.

Eco-problem solving systems have been used for practical applications, for example for the control of a manufacturing system (Sohier et al. 1996).

3 FROM GENETIC ALGORITHMS TO COMPUTATIONAL HABITATS

"I love to travel, but hate to arrive." – Albert Einstein

Some of the shortcomings of the classical GAs (brought up in Chapter 1) have been addressed in our work from the start. For example poor speciation in single populations and the lack of complex mechanisms (like cooperation between species) or the need for experimentation with various strategy parameters (like probability of mutation and crossover) every time a new application was approached. The speciation mechanisms have been of especial interest to us and in the process of studying them we have developed a data classification algorithm that displayed a world-class performance on the UCI datasets. Only later we realized that Computational Habitats may offer an environment for new approaches and solutions to most if not all those problems. However our work along the way to the point where the idea of Computational Habitats has emerged is very important and it will be detailed in this chapter.

3.1 Cooperation Mechanisms

While cooperative behaviors are appropriately studied in Echo (Holland 1992, Forrest & Jones 1994) and habitat systems, mechanisms of cooperation (for example coevolution) have also been the subject of interest and active research in the field of Genetic Algorithms. Coevolution is defined as a series of reciprocal evolutionary changes in interacting species acting as agents of selection for each other. Competitive coevolution and cooperative coevolution are two important forms of coevolutionary relationships. The nature of the relationship plays an important role in determining various components of the evolutionary model (like problem decomposition, credit assignment, etc.).

Combining competition and cooperation within a coevolution model has been used by cooperative coevolutionary GAs (Potter and De Jong, 1994). In the case of GAs, the competition is usually between individuals and not between populations. In Figure 3 we present the cooperative coevolutionary (general) model.

Figure 3: A general cooperative coevolutionary model.

Coevolution is important when EAs are applied to complex problems where modularity helps solutions to evolve in the form of interacting coadapted subcomponents (an example is behavior learning tasks, where complex behavior can be decomposed into simpler sub-behaviors). In EAs where speciation is not encouraged through special methods, the population tends to converge and the long-term preservation of diverse sub-components is practically impossible. Also, because interactions between population members are not modeled, there is no evolutionary pressure for coadaptation to occur. **Cooperative coevolution (CC)** is applicable in problems where an appropriate decomposition may be known apriori and taken advantage of. Populations of new species are introduced one by one and trained together with representatives from already trained species. The new species will try to contribute to the overall fitness by finding a special niche. When the new species meets a certain **acceptance criterion**, the older species may undergo a retraining process.

The CC GA model inherits the limitations commonly associated with a GA, like pre-determining, through experimentation, of mutation and crossover rates or the population size. A second problem encountered in this model is the quality of the contribution of a newly

introduced species (the acceptance criterion). If this contribution is not satisfactory the model proposed by Potter and De Jong (1994, 2000) simply discards the species and introduces a new one. Also, a general problem encountered by all such models is their reliance on the (user driven) choice of decomposition.

By combining a proprietary choice of decomposition with the introduction of a new algorithm called competitive-cooperative coevolutionary (CCC) GA we have obtained the best results reported in the literature to the two-spiral problem (Tulai and Oppacher 2002a) when both the weights and the architecture of the Neural Network are evolved.

Proposed first by Alexis Wieland of Mitre Corp. the two-spiral has become a favorite hard problem to solve by training neural networks.

Figure 4: Training patterns for the two-spiral problem.

The training set consists of two intertwined spirals (see Figure 4) with 97 points on each spiral for a total of 194 points. The problem is to evolve a neural network that correctly classifies all the points in the training set.

If a point that belongs to one spiral is input to the neural network the output of the network should be a positive signal and if the point belongs to the other spiral the output should be negative. When the problem is solved in this form, although it is rarely mentioned, we say that it is solved according to the 50 – 50 criterion. Sometimes (Wah and Qian, 2000) classification problems are studied using the 40 – 20 – 40 criterion.

Solutions to the two-spiral problem could either have an evolved architecture, like in the case of the cascade neural networks (CNNs) first proposed by Fahlman (Fahlman and Labiere, 1990a), or they can have a fixed architecture. The best solution to the two-spiral problem, in terms of number of hidden units (HUs), had 4 HUs and 19 connection weights and it was arrived at based on a **fixed neural network architecture** (Wah and Qian, 2000). The previously reported best solution based on an **evolved architecture** had 9 HUs and 75 weights (Fahlman and Labiere 1990b) using a second-order gradient descent method. Twelve years later we have been able to find a CNN solution with an evolved architecture with 7 HUs only (Tulai and Oppacher 2002a).

The cascade neural networks evolved through our methods are **complete** in the sense that all the characteristics of the nets, **number of hidden units, connection weights as well as neuron characteristics** are evolved. We pretty much start with 1 neuron and no weights and slowly add neurons and weights and also evolve the parameters of the neuron functions until all the points in the training set are classified.

The major change to the general form of the cooperative coevolution algorithm shown in Figure 5 was related to the introduction of new species. In our study, to alleviate the problems aforementioned, and increase the generality of the model, we introduced an extra step during which multiple populations, evolved by GAs with **different mutation and crossover rates**, participate in a competitive-cooperative coevolutionary process (competing with each other but cooperating with the previous species).

$$m = \text{size}(\text{Subpopulation } k_i) = \text{size}(\text{Population } k) / N, i=1,2,\dots,N$$

Figure 5: The initial parallel GA like setup of the CCC-GA

The use of sub-populations (also called island populations or demes) in sequential GA is similar to the fine-grain model in parallelism (Oppacher and Wineberg, 2000) and their use is linked to a very successful area of EC, the parallel Genetic Algorithms. To give just one example, Mühlenbein et al. (1991) has very successfully used parallel genetic algorithms for functions optimizations.

The newly introduced phase mimics competition between rival companies in the real world. When a new species is to be introduced we start by equally dividing the randomly initialized individuals between N sub-populations, and train each sub-population using a GA with different strategy parameters. Next, we independently train all these sub-population in conjunction with representatives (in our case, the best individuals) from the previous $k - 1$ species while monitoring their performance. The sub-population that has the worst performance a number of consecutive generations is eliminated and its individuals randomly redistributed between the remaining populations. This process mimics real life situations when employees of a company that goes bankrupt join other companies. Consequently, during the

competitive phase the competing populations may have slightly different sizes. The size of the population of the winning species will be equal to the sum of all initial populations but it is clear that the mutation and crossover rates are not known apriori as they depend on the GA used by the winning population. The winning species thus joins other previous winners in the cooperative coevolutionary phase of the algorithm. This process is repeated for every hidden unit introduced in the CNN.

The purpose of the competitive phase is twofold. On the one hand it evolves a new species and on the other hand it selects the GA with the most appropriate mutation and crossover rates to evolve this new species. The purpose of the cooperative coevolutionary phase is to find those representatives from each species that together provide the best solution to the problem. The GA based method that we have introduced has also been compared to CC using Evolutionary Strategies (Schwefel, 1995).

Figure 6: The average decay of the SES as a function of the number of hidden units.

The CC-ES method lacks the competitive phase of the CCC-GA but uses a pool of 8 species and selects the one making the best contribution to introduce as a new species.

As can be seen in Figure 6, the CCC-GA method and the CC-ES method have very similar behaviors when it comes to decreasing the Squared Error Sum (SES), the two curves being almost identical when the results of 50 runs were averaged. Both CCC-GA and CC-ES methods produced better results than the cascade correlation learning (gradient descent based) method (Fahlman & Labiere 1990a,1990b) , both in terms of the average number of hidden units as well as the minimum number of hidden units required for a cascade network solution to the two-spiral problem. However the cascade correlation learning method is a much faster method compared to both coevolutionary methods.

Table 2: Required number of hidden units for the three methods under study

Method	Hidden Units				Time
	Min	Max	Mean	Failed	[s]
CCC-GA	7	16	10.54± 0.71	0	1620
CC-ES	8	15	9.9± 0.57	0	2700
Cascade correlation	10	19	13.4± 0.04	7	5

The CC-ES algorithm has produced the best results in terms of average number of hidden units but the CCC-GA produced a cascade network with only 7 hidden units. In fact the neural network with 7 hidden units is the **smallest evolved CNN solution** (the number of hidden units is not known apriori) reported in the literature.

Table 2 shows minimum and maximum number of hidden units produced by each method as well as the average number of hidden units produced by each method (50 runs for the CCC-GA and CC-ES and 10000 runs for the cascade correlation method), along with 99-percent

confidence on the mean. Although they show similar results the CCC-GA method is faster than the CCC-ES method.

As part of our research we have also studied various methods of retraining the species. Both the CCC-GA and CC-ES algorithms use retraining as an important method for further reducing the SES at the network output. For example, when no retraining was used by the CCC-GA method, the performance decreased so severely that only 2 runs were successful in 10 runs performed. This is a very important aspect because of its artificial nature. In a natural habitat environment various cooperating species coexist and evolve together naturally and this kind of behavior would also be expected in an artificial environment like the Computational Habitats.

Figure 7: The improvements in squared error sum (SES) after new hidden units, retraining and the sum of the two.

Contribution of retraining to the quality of the solution is also suggested by Figure 7 where the improvement in SES due to retraining is displayed together with the improvement obtained by introducing new HUs.

For our study we used 3 different methods of retraining; random, (0 to maxHU-1) or the reverse of that (maxHU-1 to 0). The results of the 50 different runs performed with the CCC-GA algorithm are presented in Table 3.

Table 3: Required number of hidden units for CCC-GA when different retraining methods are used

Method	Hidden Units			
	Min	Max	Mean	Failed
random	7	16	10.54 ± 0.71	0
$0 \rightarrow \max HU - 1$	8	16	10.42 ± 0.58	0
$\max HU - 1 \rightarrow 0$	8	14	10.46 ± 0.4	0

While the average number of hidden units seems to be similar for the three methods it does appear that the random retraining yields a slightly higher variance of the results. This factor may have favoured the discovery of the cascade network with 7 hidden units.

We have also studied the robustness of the CCC-GA method. While the CC-ES chooses from a pool of 8 different species (why not 4 or 10?), the CCC-GA method doesn't have to make such a choice. In fact, if the CC-ES model doesn't use a pool of species and uses a randomly initialized population, the results (for example the average number of hidden units per evolved cascade network) degrade very significantly. From this point of view it is clear that the CCC-GA method is more robust than the CC-ES method.

When it comes to the evolutionary operators used, ES does not use crossover and the mutation probabilities are independently evolved by the algorithm (so the user has only to initialize an initial vector). While this seems to introduce a clear advantage for the CC-ES method, the CCC-GA compensates for it by evolving the species with the crossover and mutation probabilities best suited for a certain phase of the algorithm. In addition to that, through experimentation we found out that if the ES initial choice is to evolve the mutation

probabilities as a group rather than individually (each mutation step evolved independently) the results of the CC-ES will again degrade from the results in Table 3.

3.2 Evolving GA Strategy Parameters

Evolutionary Algorithms are increasingly used, as an alternative to analytical methods, to solve problems made intractable by their complexity and high number of variables. When compared in speed to more traditional methods, EAs are considered slow, but with the power of computers growing exponentially, this becomes less and less of an issue.

An important criterion when choosing an algorithm is its ease of use. Algorithms that require fewer parameters to set are preferable, especially when there are no clear rules for setting them. This criterion may sometimes be given more weight over other criteria like speed or even accuracy. One complaint that could be often heard from first time users of Genetic Algorithms is the need to experimentally determine the probabilities of the genetic operators. In spite of many years of research in this area there is still no theoretical support for using certain genetic operators or for setting the associated probabilities, for a given problem. Algorithms that incorporate methods to isolate the user from these decisions are of particular interest and the GA community has investigated ways to do away with these parameters (Harik and Lobo 1999, Lobo and Goldberg 2001). Our first contribution to these efforts has been reported in the previous section when we mentioned that during the competitive phase of the CCC-GA algorithm each sub-population has been trained by a GA with different strategy parameters. For the two-spiral problem we have used 16 sub-populations every time a new species was introduced and if we designate the mutation and crossover probability used by a sub-population by $(\mu_i, \chi_i), i = 1, 16$, the points will be uniformly distributed over the probability space as shown in Figure 8.

Figure 8: The distribution of the genetic operators' probabilities for 16 sub-populations

Although a step forward from the classical approach of using a fixed mutation and crossover probability, the 16 points represent only a rough coverage of the 2-D probability space and this setup is run only when the new species is introduced. However, when the species are retrained the probabilities will not change and there is no guarantee the pair of probabilities used (for mutation and crossover) would continue to be the best choice for that particular stage of the algorithm.

In a subsequent research we have generalized the competitive phase of CCC-GA and rather than assigning a fixed pair of mutation and crossover probability to each sub-population we have encoded these probabilities as chromosomes in each individual, initialize them randomly and let them freely evolve together with the application chromosomes (Tulai and Oppacher 2002b).

As shown in Figure 9, the genetic representation of an individual, the genome, is composed of n application chromosomes and 4 strategy chromosomes.

x_1	..	x_n	p_c	p_m	p_r	s
-------	----	-------	-------	-------	-------	-----

Figure 9: Genome representation: $\{x_1, \dots, x_n\}$ are application dependent chromosomes and $\{p_c, p_m, p_r, s\}$ are chromosomes representing the GA strategy parameters.

This GA uses three genetic operators: crossover, mutation and recombination similar to the approach taken by Mühlenbein et al. (1993). In addition to the three genetic operators, an evolvable scaling factor used to alter the mutation steps is also included. This approach has been applied in ES with excellent results (e.g. Schwefel 1995). Our intention was to trade some efficiency and speed for a more general, easier to use, generally applicable algorithm.

The four chromosomes representing the GA strategy parameters were mutated first with probability 1 (they are always mutated) using the relations given in equation (3.1).

$$p_i(t+1) = p_i(t) + \text{Gauss}(0, sd_i), \quad p_i \in [L_i, U_i], \quad p_i \in \{p_c, p_m, p_r, s\} \quad (3.1)$$

In all the practical studies the standard deviations used in mutating the three genetic operator probabilities, sd_i , $i = 1, \dots, 3$ have been set to 0.1.

The application chromosomes $x_i(t)$ are updated with probability $p_m(t+1)$ using the equation,

$$x_i(t+1) = x_i(t) + \frac{(U_i - L_i)}{2} * \text{Gauss}(0, s(t+1)), \quad i \in \{1, \dots, n\}, \quad x_i \in [L_i, U_i] \quad (3.2)$$

For all our studies, we use a GA with overlapping populations with 50% probability of replacement and Tournament selection.

In applications to which the cooperative coevolution model is not applicable we do not need to introduce new species and therefore the CCC-GA as a whole may not be applicable. Fortunately the competitive phase of the CCC-GA can be used independently. It starts as a

parallel genetic algorithm and through consecutive eliminations of underperforming populations evolves towards a single population and the GA used for evolving this population.

We evaluated the GA with strategy parameters encoded as chromosomes on a test bed of seven continuous functions (that had to be optimized) and we have also used the generalized CCC-GA on the more difficult task of evolving cascade neural networks for solving the two-spiral classification problem (for the purpose of comparing it to the previously reported results).

The seven functions test bed that we used is presented in Table 4. The first two functions, the spheroid and the hyper-ellipsoid, have been chosen for a study of the convergence performance when the global minimum is sought with a high degree of accuracy (10^{-10}). Functions F_3 - F_7 are known by the name of those who proposed them and, in their case the termination criterion is reaching the global minimum with at least 3 digits accuracy.

Figure 10: Evolution of the population fitness for the sphere and the weighted sphere model (hyper-ellipsoid) for $n=32$

Table 4: Seven functions test bed.

No	Function	Limits
1	$f_1(x) = \sum_{i=1}^{32} x_i^2$	$-5.12 \leq x_i \leq 5.12$
2	$f_2(x) = \sum_{i=1}^{32} ix_i^2$	$-5.12 \leq x_i \leq 5.12$
3	$f_3(x) = nA + \sum_{i=1}^n (x_i^2 - A \cos(2\pi x_i)) \quad A = 10$	$-5.12 \leq x_i \leq 5.12$
4	$f_4(x) = nV - \sum_{i=1}^n x_i \sin(\sqrt{ x_i }) \quad V = 418.9829$	$-500 \leq x_i \leq 500$
5	$f_5(x) = 1 + \sum_{i=1}^n x_i^2 / 4000 - \prod_{i=1}^n \cos(\frac{x_i}{\sqrt{i}})$	$-600 \leq x_i \leq 600$
6	$f_6(x) = 20 + e - 20e^{-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}} - e^{\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)}$	$-30 \leq x_i \leq 30$
7	$f_7(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$	$-2.048 \leq x_i \leq 2.048$

One remarkable behavior of the GA with self-evolving strategy parameters is its robustness. Very few changes were required to the parameters between experiments. As it can be seen in Table 5 the only differences in the parameters used by the algorithm for the continuous function optimization task are related to the scaling factor and are primarily caused by the difference in accuracy required for the first two functions optimized.

The results obtained for the first two functions are presented in Figure 10 and it is interesting to notice that with a simple, nature derived algorithm, we obtained results closely resembling Voigt's results using soft recombination and a population of 512 individuals (Voigt 1995).

For functions F3-F7, the results are presented in Table 6.

Table 5: Algorithm parameters for function optimization and CNNs evolving tasks.

Parameter	Function optimization		Cascade Neural Net
	F ₁ -F ₂	F ₃ -F ₇	
Number of initial subpopulations (λ)	8		16
Subpopulation size (μ)	32		64
No. of times a subpopulation should rank last before re-distribution (L)	4		10
Std. deviations (sd) used for updating the genetic operators probabilities	{0.1, 0.1, 0.1}		
Std. deviation used for updating the scaling chromosome (sd _s)	0.0001		0.1
Scaling factor limits (s)	[10 ⁻⁷ , 1-10 ⁻⁷]		[0.01, 0.99]
Global minimum required accuracy	10 ⁻¹⁰	10 ⁻³	N/A

The results presented in Table 6 with this algorithm are better than those reported in (Potter & De Jong 1994) (a precise comparison was not possible because in their paper the results are graphically presented) using cooperative co-evolution. Where a comparison was possible, the results obtained with more sophisticated PGAs (specifically targeting function optimization problems as in Mühlenbein et al. 1991, 1993) were better than those shown in Table 6. However, while (Mühlenbein et al. 1991) reports that for F₄, for example, the global optimum has not been found in 4 of the 50 runs performed, our algorithm showed robustness in finding the global optimum in all 100 runs performed for all the functions studied.

Table 6: Results for functions F_3 - F_7 .

F	n	Function evaluations		
		Best	Worst	Average (100)
F_3	10	4968	11835	8052
F_3	20	7836	22207	15405
F_3	50	30292	80468	48073
F_4	10	18712	68074	23935
F_5^*	10	3696	5862	4506
F_6	30	21952	48940	36545
F_7	2	3954	47916	10491

We have also used the generalized CCC-GA algorithm for evolving cascade neural networks (CNNs) for the two-spiral problem and compared the results to those obtained with the initial version of the CCC-GA algorithm.

Table 7 shows the minimum, maximum and the average number of hidden units of the evolved CNNs, along with 99-percent confidence on the mean, over 100 runs. As expected the new algorithm is about 25% slower than the old algorithm as additional computations are associated with updating the GA parameters. However, in spite of its improved generality, the new algorithm performs as well as the old one for the very complicated task of evolving cascade neural networks.

Table 7: The evolved CNN length (in hidden units) for the two GAs under comparison

Method	Hidden Units			Time	
	Min	Max	Mean	Failed	[min]
Old CCC-GA	7	16	10.43 ± 0.476	0	27
New CCC-GA	8	15	10.42 ± 0.346	0	34

The goal of our research was to create EAs with fewer adjustable parameters that could be applied successfully to a large range of problems. In Tulai and Oppacher (2002b), we introduce a **generalized version** of a parallel GA that mimics the real life workforce migration following the collapse of a company (Tulai and Oppacher 2002a). The generalization was achieved by encoding the strategy parameters as chromosomes in every genome in the population, and having them evolve together with the application chromosomes.

The original CCC-GA algorithm was shown to be robust to population variations when used for evolving cascade neural networks. The generalized CCC-GA algorithm not only has less parameters but most of them could keep their original values for all the problems studied independent of the accuracy sought, further enhancing its original robustness (Tulai and Oppacher 2002b).

3.3 MSWV - Exploiting Speciation in Data Classification

Speciation is generally considered to be the mechanism of differentiation of individuals to exploit environmental niches. Our interest in speciation has translated into the development of a data classification algorithm (Tulai & Oppacher 2004a, Tulai & Oppacher 2004b, Tulai & Oppacher 2004c, and Tulai & Oppacher 2004 d) with world-class results on the UCI datasets.

Data classification, as a supervised learning task, has been one of the most researched subjects and the progress in this area has translated into a large variety of supervised learning

algorithms. Among the genetics-based machine learning (GBML) systems, the more successful ones have been the learning classifier systems (LCS) with XCS being the main example (Wilson 1995, Wilson 1998). While the Multiple Species Weighted Voting (MSWV) algorithm qualifies as a GBML because of its use of populations of individuals and two genetic operators, it is not an LCS.

The use of N 2-class classifiers in N -class classification tasks, that MSWV uses (discovered independently), is also the standard method used by Support Vector Machines (Vapnik 1998). There are also other algorithms using $N(N-1)/2$ SVM classifiers (one for each pair of classes) for an N -class classification task (Friedman 1996).

In section 3.3.1 we introduce a terminology and present a background of the classification theory while in section 3.3.2 we present the important techniques used by the algorithm. In section 3.3.3 we present the experimental results and compare them with well known competing techniques. In section 3.3.4 we describe in detail the automatic speciation mechanism followed by conclusions in section 3.3.6.

3.3.1 Background

A dataset D is a subset of $\mathcal{X} \times \mathcal{C}$ where $\mathcal{X}$ is the input space, also called the feature space, and $\mathcal{C}$ is the class space. In general $\mathcal{X} = X_1 \times \dots \times X_n$ but in many practical applications $X_i = \mathbb{R}$ so $\mathcal{X} = \mathbb{R}^n$ and the feature space is n -dimensional. $\mathcal{C} = \{c_1, \dots, c_N\}$, usually a discrete set of small cardinality N , is the class space. A record (instance or sample) of a data set, is a point $p = \{\mathbf{x}, c_i\} = \{x_1, \dots, x_n, c_i\}$ where $\{x_j\}_{j=1,n}$ are the feature values and c_i is the class of the record (sometimes called the record tag).

Classification is the problem of predicting the value of an output variable $y \in \mathcal{C}$ based on a given value of the input variable $\mathbf{x} = \{x_1, \dots, x_n\}$.

An attribute is a Boolean valued function $h_i : X_i \rightarrow \{0,1\}$. A decision rule r is a function $r : \{0,1\}^n \rightarrow \mathcal{C}$ usually built as a conjunction or disjunction of attributes. Decision rules are

used in classification and they are usually constructed by a learning algorithm from a training set of L points $p_i = \{x_i, y_i\}, i = 1, L$ of known input and output values. For a given point $\{x, c_i\}$ with $x = \{x_1, \dots, x_n\}$ a correct classification occurs when $r(\{h_1(x_1), \dots, h_n(x_n)\}) = \hat{y}(x) = c_i$, otherwise ($\hat{y}(x) \neq c_i$) it is considered an error. MSWV uses decision trees with internal nodes storing Boolean valued attributes and the leaf nodes containing a tag. Because the path from the root node to a leaf node is a conjunction of Boolean attributes, a decision tree can be seen as a disjunction of a limited number of conjunctions.

In general almost all learning algorithms are procedures for obtaining estimates $\{\hat{f}_i(x)\}_1^N$ of the set of conditional probabilities:

$$\{f_i(x) = \Pr(y = c_i | x)\}_1^N \quad (3.3)$$

There are two methods for obtaining these probability estimates (Friedman 1996), the density estimation approach and a regression methodology. The density estimation approach uses Bayes' theorem. A well known example of a classification algorithm using the density estimation paradigm is Naïve Bayes (John & Langley 1995) but as referenced in (Friedman 1996) there are many others.

The second approach attempts to directly estimate the conditional probabilities (3.3). Examples of algorithms using the regression methodology are the decision tree induction methods (Frank & Witten 1998, Quinlan 1993), the nearest neighbor methods (Aha et. al. 1991) and others. MSWV also belongs to this class of algorithms. In the case of the algorithms using decision tree representations, f would be nothing else but a function built on the Boolean valued attributes $h(x)$ (like a decision rule). MSWV uses a GA to modify these decision trees and search for those trees that contain the best decision rules r giving us the best probability estimates $\{\hat{f}_i(x)\}_1^N$.

The predicted class $\hat{i}$ may be obtained from the probability estimates using:

$$\hat{i}(\mathbf{x}) = \arg \max_{1 \leq i \leq N} L_i \hat{f}_i(\mathbf{x}) \quad (3.4)$$

Very often $L_i = 1$ (Friedman 1996) which is equivalent to assigning the predicted class to be the most probable for a given $\mathbf{x}$. In section 3.3.2.7, the equivalent of equation (3.4) will be given by equation (3.8) showing how the decision is made in the case of MSWV.

3.3.2 Algorithm Characteristics

In this section we detail the main characteristics of the MSWV system.

3.3.2.1 Representation and Species Definition

MSWV uses populations of individuals to create 2-class classifiers. The genotype and phenotype of an individual is a decision tree with internal nodes carrying Boolean valued attributes and leaf nodes carrying one of two possible class labels. What distinguishes MSWV from other algorithms using decision trees is the constraint we impose that each internal node must carry a Boolean valued attribute constructed from a *feature distinct* from the other internal nodes in the tree. This restriction implies that a decision tree can have a maximum of internal n nodes (where n , following the terminology in section 2, is the number of features) and a maximum of $n + 1$ leaf nodes. Because the path from the root node to a leaf node represents a decision rule (as a conjunction of Boolean valued attributes) each decision tree can carry a maximum of $n + 1$ decision rules. This restriction significantly reduces the size of the search space.

The decision trees are randomly initialized. Each internal node stores a Boolean valued attribute as a triplet (*feature#*, *featuretype*, *values*). The possible features and the attribute functions built on them are:

Boolean features (type 1). $X_i = \{a, b\}$. An internal node stores $(i, 1, v)$ with $v \in \{a, b\}$, $h_i(x_i) = 1 \Leftrightarrow x_i = v$.

Set features (type 2). $X_i = \{x_i^1, \dots, x_i^k\}$. An internal node stores $(i, 2, A)$ with $A \subseteq X_i$, $h_i(x_i^j) = 1 \Leftrightarrow x_i^j \in A$

Numerical features (Integer type 3, Floating point type 4). $X_i = [x_i^{\min}, x_i^{\max}]$, $x_i \in \mathbb{Z} \mid \mathbb{R}$. A node stores $(i, 3 \mid 4, \{v_1, v_2\})$, $[v_1, v_2] \subseteq X_i$, $h_i(x_i) = 1 \Leftrightarrow x_i \in [v_1, v_2]$

Each 2-class classifier is represented by a population of decision trees with leaf nodes carrying one of two possible class labels, either a class $c_i \in \mathcal{C}$ or the "don't know" class that we tag as "?" (i.e. $? = \mathcal{C} - \{c_i\}$). A population of individuals recognizing the same two classes $\{c_i, ?\}$ forms a species, so each species is a 2-class classifier. For a N-class classification task, MSWV creates N species.

The training is done using the 1-v-r (one-versus-rest) method. For a given training set T the N classes induce a partition $T = T_1 \cup \dots \cup T_N$ with T_i containing only the records tagged with c_i . The i -th species (corresponding to the i -th 2-class classifier) is trained with the examples in T_i as positive examples and all other examples ($T - T_i$) as negative examples (treated as the "don't know" class).

The size of the i -th species population is automatically calculated with the formula:

$$m_i = \begin{cases} 20 & \text{if } \text{card}(T_i)/2 < 20 \\ 200 & \text{if } \text{card}(T_i)/2 > 200 \\ \text{card}(T_i)/2 & \text{otherwise} \end{cases} \quad (3.5)$$

The reason behind using a population size of $\text{card}(T_i)/2$ is linked to the assumption that in the worst case the training subset T_i may contain just 2 examples for each possible classification rule that needs to be discovered in which case the maximum number of rules the i -th species should discover (although there is no guarantee they will be discovered) is $\text{card}(T_i)/2$. The upper limit of 200 on the population size is imposed by computational

considerations. The lower limit of 20 is imposed by training time considerations and will be discussed in section 3.3.2.6.

3.3.2.2 Mutation Operators

In its current version MSWV uses only mutation and selection (described in the next section). Every generation all the individuals of a species (the species could be trained independently and in parallel) are mutated with probability 1, followed by selection. MSWV implements 6 mutation operators but with equal probability uses only one when mutation is applied. The six mutation operators are shown in Table 8. The term "randomly" refers to a uniform distribution random variable.

Table 8: The mutation operators used by MSWV.

<i>Operator</i>	<i>Description</i>
Add node	Adds a new internal node (and a new attribute)
Remove node	Randomly selects a leaf node and removes it and its parent node
Swap two leaves	Randomly selects a pair of sibling leaves and swaps them.
Swap internal nodes	Randomly selects two internal nodes and swaps them
Swap two branches	Randomly selects two internal nodes and swaps the sub-trees headed by them
Attribute change	<p><i>Boolean features:</i> the value used in the node is replaced by its complement</p> <p><i>Set features:</i> With equal probability either adds one element to the set stored in the node or removes one</p> <p><i>Numerical features:</i> Randomly either shift the range up down or shrink expand the range by a random gain</p>

3.3.2.3 Rule Discovery and Automatic Speciation

One can classify the decision rules into *weak* and *strong* rules depending on whether they cover a small or a large number of training examples. There are datasets governed by one (or a very small number of) strong classification rule(s) in which case placing all the individuals in one large population and having them compete with each other by applying the selection operator

seems to be the right way to go. Copies of the fittest individual will propagate through the population of decision trees resulting in an accelerated search for the strongest classification rule.

On the other hand, in general, the training examples are covered by a mixture of strong and weak rules (Carvalho & Freitas 2002). This would suggest that following a simple hill climbing strategy by applying selection only between a parent and its offspring (we call this **restricted selection**) while slowing down the process of rule discovery, would ensure that both the strong and the weak rules were pursued by the evolving population.

MSWV achieves a compromise between the two extreme cases. Initially, each species starts with a population of hill climbers, individuals that mutate and use restricted selection to maintain the diversity of the evolved rules. Periodically the algorithm checks for **similar** individuals and places them together forming subspecies. The similarity is defined not in terms of genetic makeup but in terms of proficiency in classifying positive and negative examples. The speciation process is formally described as follows. Let S_i represent the set of all individuals in species i , trained with positive examples from the set T_i and negative examples from the set $T - T_i$. We introduce a pre-order relation on the elements of S_i . If $s_1, s_2 \in S_i$ are two individuals from species i and by $s_1(T_i) \subseteq T_i$, $s_1(T - T_i) \subseteq T - T_i$, $s_2(T_i) \subseteq T_i$ and $s_2(T - T_i) \subseteq T - T_i$ we denote the sets of positive and negative examples correctly classified by s_1 and s_2 we say that

$$s_1 \geq s_2 \Leftrightarrow s_1(T_i) \supseteq s_2(T_i) \ \& \ s_1(T - T_i) \supseteq s_2(T - T_i) \quad (3.6)$$

In other words, an individual is greater than or equal to another individual (from the same species) if and only if the first individual can correctly classify all positive and all negative training examples the second individual classifies. It can be easily shown that the relation " $\geq$ " is reflexive and transitive but not antisymmetric. MSWV uses the pre-order relationship on the elements of S_i to create subspecies of individuals that are allowed to compete with each other

and accelerate the search for specific decision rules. Because it is computationally intensive this operation (that we call **automatic speciation**) is not performed every generation but at regular intervals. All the experiments described in section 4 have used a speciation period of 50 generations. When the speciation operation is performed the second (or subsequent) time subspecies may already exist in the population. In this case only the fittest individuals from each subspecies participate in the speciation process in which case the pre-order relation between individuals will result in mergers between their respective subspecies. The use of the fittest individual of a subspecies is justified by the lack of diversity that characterizes the subspecies, as a result of full selection pressure. We would like to point out that individuals of a subspecies may also be allowed to mate if an effective crossover operator is devised in the future.

3.3.2.4 Rule Confidence and Rule Weight

As previously mentioned, a decision tree encodes as many decision rules as there are leaf nodes in the tree. Although not mentioned in section 3.3.2.1, associated with each leaf there is some additional information built during the learning phase.

- A counter P that counts all the positive examples that have exercised the tree path (i.e. decision rule) ending at this leaf.
- A counter N that counts all the negative examples that have exercised the tree path ending at this leaf.
- A rule confidence coefficient calculated, following training, as follows:

$$\mu = \frac{P}{P + N} \text{ (also called the true positive rate) for leaves tagged with } c_i$$

$$\mu = \frac{N}{P + N} \text{ (also called the true negative rate) for leaves tagged with ?}$$

The rule confidence coefficients are used to weight the individual vote during the testing phase (see section 3.3.2.7) and remove the need for post pruning.

Datasets often contain instances with unknown values, and learning algorithms must be able to deal with them with minimal degradation of performance.

MSWV treats the unknown (missing) values similar but not identical to other practical learning schemes like C4.5 (Quinlan 1993) and PART (Frank & Witten 1998). When a record is evaluated (during training or testing) a path weight variable pW , initially set to 1, is associated with the path taken by the record through the decision tree. Associated with each internal node of a decision tree there are 4 variables (p_l, p_r, n_l, n_r) counting how many positive or negative examples have gone left or right. If an instance cannot be assigned deterministically to a branch because of an unknown value we assume that it may go on both branches and consequently two paths are created from that node down. The path weights of these two paths are updated using the 4 variables stored in the node with separate equations depending on whether we are testing or training.

During testing, the left and right path weight are given by $pW \cdot \rho_l$ and $pW \cdot \rho_r$ where

$$\rho_l = (p_l + n_l) / (p_l + p_r + n_l + n_r), \rho_r = (p_r + n_r) / (p_l + p_r + n_l + n_r)$$

During training after calculating the path weight updating factors ρ_l and ρ_r we also have to (on-line) update the variables (p_l, p_r) or (n_l, n_r) :

$$\begin{aligned} \rho_l &= p_l / (p_l + p_r), \rho_r = p_r / (p_l + p_r), p_l = p_l + pW_l, p_r = p_r + pW_r \\ \rho_l &= n_l / (n_l + n_r), \rho_r = n_r / (n_l + n_r), n_l = n_l + pW_l, n_r = n_r + pW_r \end{aligned}$$

Because a path from the root node to a leaf node defines a rule, a path weight is nothing else but a rule weight. In section 3.3.2.5 we show how the path weights (or rule weights) are used during training (in fitness calculation) and in section 3.3.2.7 we show how they are used during testing (in vote weighting).

3.3.2.5 Individual Fitness Calculation

The fitness of an individual is calculated only once, when the individual is born, using the training data set. When during the process of evaluating a decision tree shaped chromosome on a training example a leaf node is reached the following updates take place. If the leaf has a

tag c_i that matches the class of the example we increment the positive examples counter P and give a reward, otherwise we increment the negative examples counter N and give no reward. If the leaf is tagged "?" and the example is tagged c_i we increment the positive examples counter P and give no reward, otherwise we increment the negative examples counter N and give a reward.

The reward score for correctly classifying an example depends on the path weight. However we need to correct for the imbalance that may exist in the data set. If a class has very few representatives we need to give a higher reward for correctly classifying samples from this class. Consequently, the reward for correctly classifying a positive example $x \in T_i \subset T$ is

$$r_p(x) = \sum_{\text{all paths}} p W \cdot \text{card}(T) / \text{card}(T_i). \text{ The reward for classifying a negative example}$$

$x \in T - T_i$ (at a leaf node tagged with the unknown class "?") is given by the equation:

$$r_n(x) = \sum_{\text{all paths}} p W \cdot \text{card}(T) / \text{card}(T - T_i). \text{ The sum over all paths is required for}$$

handling unknown values as explained in section 3.4. With these definitions, the fitness of an individual $s \in S_i$ is given by: $g_i(s) = (\sum_{x \in T_i} r_p(x) + \sum_{x \in T - T_i} r_n(x)) / 2$ (therefore

$$g_i(s) \in [0, \text{card}(T)]).$$

One of the major concerns in machine learning is the generalization power of the rules learned. To encourage the emergence of short rules with a high generalization potential, MSWV reduces an individual's fitness by a penalty factor defined as: $\delta = 0.0005(n - 1)L$ where n is the number of internal nodes in the tree (and distinct features used in the decision tree), L is the number of records in the training data set and the constant 0.0005 has been experimentally determined. Assuming a training dataset of 1000 instances, this small penalty factor implies that adding a new internal node to a decision tree must be justified by an increase in fitness greater than 0.5.

3.3.2.6 Training Time

Unlike other GBML systems that use a fixed, experimentally determined training time parameter (Bernadó-Mansilla & Garrell-Guiu 2003, Carvalho & Freitas 2002), MSWV uses a dynamic stopping criterion. The reasons why a stopping criterion is superior to a fixed training time method are:

- if the training phase stops too early the achieved rule accuracy may be too low
- a training phase that is too long may encourage data overfitting
- the "optimum" training time may greatly vary from dataset to dataset
- it improves the algorithm efficiency (stopped species do not consume CPU time)

Every Δ generations MSWV calculates the average fitness of all the individuals of a species. If the average fitness of all individuals of species i at time $t = j\Delta$ is given by $\bar{g}_i(j\Delta)$, the training will stop at time $j\Delta$ if the following condition is met

$$\frac{\bar{g}_i(j\Delta) - \bar{g}_i((j-3)\Delta)}{3} \leq 0.02 \max_{k \geq 1} (\bar{g}_i(k\Delta) - \bar{g}_i((k-1)\Delta)) \quad (3.7)$$

MSWV system uses a value of 10 for Δ . Put in words, a species is evolved until the increase in the (smoothed over 2Δ) average fitness of its individuals falls below 2% of the highest jump in average fitness over Δ generations. Because the highest jump in fitness usually occurs in the first Δ generations for almost all datasets it is reasonable to ensure the same behavior in the case of those datasets that have a very small number of examples for a certain class (these datasets are very rare but they do exist in the real-world). It is because of these special cases that we established a lower limit on population of 20 (see section 3.3.2.1) to increase the probability of positive evolutionary changes and a non zero average fitness increase early in the training.

3.3.2.7 Testing

Given an instance $\mathbf{x}$ from a testing dataset with N classes, MSWV uses a voting scheme to predict its class. Because the individuals in a subspecies are all very similar (due to the strong selection pressure) only the fittest individual in the subspecies is allowed to vote. Experiments performed with a very large number of datasets have shown there is no difference in results if all are allowed to vote or only the best is allowed to vote (but this method is more efficient). Let's assume species i finishes the training phase, with its population distributed in M_i subspecies. Let's also denote by M the maximum number of subspecies in any of the N species (i.e. $M = \max_{i=1,N} \text{card}(M_i)$). The outcome of the weighted voting process is given by the following formula (also see equation (3.4) for reference):

$$\hat{i}(\mathbf{x}) = \arg \max_{1 \leq i \leq N} \frac{M}{\text{card}(M_i)} \sum_{j=1}^{M_i} v_{ij}(\mathbf{x}) \quad (3.8)$$

where $v_{ij}(\mathbf{x})$ is the confidence and path (rule) weighted vote of the fittest individual from the j -th subspecies of the i -th species and is given by the formula:

$$v_{ij}(\mathbf{x}) = \sum_{\text{all paths}} (-1)^{l(\text{leafTag} == "n")} \mu \cdot p W, l(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} = \text{true} \\ 0 & \text{if } \mathbf{x} = \text{false} \end{cases} \quad (3.9)$$

where μ is the confidence coefficient calculated as explained in section 3.3.2.4, $p W$ is the path weight and the summation over all possible paths is required for handling instances with unknowns. Confidence weighting removes the need for post pruning (for example, the vote of a rule with confidence 0, is 0).

3.3.3 Experimental Results

The performance of MSWV has been evaluated on a set of thirty-six standard datasets available from UCI at: www.ics.uci.edu/~mllearn/MLRepository.html. The thirty-six datasets used for the study exhibit a large variety of characteristics in terms of number of instances,

number and diversity of features, number of classes, missing feature values, etc. The datasets and their characteristics are listed in Table 9.

Table 9: The datasets used for experiments
(unknown values given as a percentage of total).

Dataset	Id	Inst	Unknown	#feat	#nom	#num	#class
Audiology	aud	226	2.0	69	69		24
Balance	bal	625	-	4		4	3
Breast-w	brw	699	0.3	9		9	2
Bupa	bpa	345	-	6		6	2
Contraceptive	cmc	1473	-	9	4	5	3
Credit-austral.	cra	690	0.6	15	9	6	2
Credit-german	crg	1000	-	20	13	7	2
Glass	gls	214	-	9		9	6
Heart-c	h-c	303	0.2	13	7	6	2
Heart-h	h-h	294	20.5	13	7	6	2
Hepatitis	hep	155	5.6	19	12	7	2
Ionosphere	ion	351	-	34		34	2
Iris	irs	150	-	4		4	3
Labor	lbr	57	35.7	16	8	8	2
Pima-indians	pmi	768	-	8		8	2
Primary-tumor	prt	339	3.9	17	17		22
Sonar	snr	208	-	60		60	2
Soybean	soy	683	9.8	35	35		19
Vehicle	veh	846	-	18		18	4
Vote	vot	435	5.6	16	16		2
Vowel	vow	990	-	13	3	10	10
Wine	wne	178	-	13		13	3
Yeast	yst	1484	-	8		8	10
Zoo	zoo	101	-	16	16		7
Adult	ADL	48842	0.9	14	8	6	2
Hypothyroid	HTH	3772	5.5	29	22	7	4
Kr-vs-kp	KRK	3196	-	36	36		2
Led (10%noise)	LED	6000	-	7	7		10
Letter	LTT	20000	-	16		16	26
Mushrooms	MUS	8124	1.4	22	22		2
Optical-digits	ODG	5620	-	64		64	10
Satellite-image	SAT	6435	-	36		36	6
Segment	SEG	2310	-	19		19	7
Sick	SIC	3772	5.5	29	22	7	2

Splice	SPL	3190	-	60	60	3
Waveform+noise	WVF	5000	-	40	40	3

The performance of MSWV is compared to six other learning algorithms: NB (John & Langley 1995), IB1 and IB3 (IBk or Instance-Based learner, Aha et. al. 1991, that we use with $k=1,3$), C4.5 (revision 8, an induction tree algorithm, Quinlan 1993), PART (an algorithm for inferring rules by repeatedly generating partial decision trees, Frank & Witten 1998) and SMO (Sequential Minimal Optimization, a SVM classifier system, Platt 1998), all of them run using the Weka v3.4 package (Witten & Frank 2000) available at: www.cs.waikato.ac.nz/ml/weka.

The datasets are divided in two groups, twenty-four small (under 2000 instances) datasets and twelve large (over 2000 instances) datasets.

On the group of small datasets (lower case IDs) the comparison between classifiers is based on averaging ten ten-fold cross-validation runs. To test the statistical significance of the differences between classifiers on this group of datasets we used a paired two-sided t -test at 99% confidence level.

On the group of large datasets (upper case IDs) the comparison is based on the holdout estimate, where 33.3% of the instances are used for training and 66.6% are used for testing. The statistical significance of the differences between classifiers is performed using a test for the differences of two proportions at 99% confidence level. The large datasets have been shuffled offline and we have ensured that all classification systems have used the same partitions for training and testing.

Finally, the paired Wilcoxon signed rank test is used to calculate the statistical significance of the overall observed differences between two learning systems.

The accuracy rates (given as percentage of correct classifications) of the classification systems are summarized in Table 10. Results for the six learning systems used in the study are marked with $\circ$ if they show significant improvement over the corresponding results for MSWV, and with $\bullet$ if they show significant degradation.

The results presented in Table 10 can serve as basis for several observations. The first observation is that on this set of thirty-six datasets MSWV significantly outperforms 4 other classification systems (NB, IB1, C4.5 and PART), slightly outperforms IB3 and it is outperformed by SMO. This conclusion is based on the results presented in the row labeled W-L-T (wins-losses-ties) and also from the outcome of the paired Wilcoxon signed rank test as shown in the row labeled "Confidence". For example, MSWV significantly outperforms C4.5 on 17 datasets and it is significantly outperformed by C4.5 on 7 datasets. Using the results of the Wilcoxon test we can say that on this group of datasets MSWV improves C4.5 with 97.6% confidence but it is also improved by SMO with 54.7% confidence. The comparison with the other classification systems is done similarly.

Using a similar methodology and the experimental results on thirty-six datasets, (Bernadó-Mansilla & Garrell-Guiu 2003) reported that XCS and their own UCS (both are Learning Classifier Systems) were outperformed by C4.5, PART and IB3. The SMO classifier used in our study could handle nominal and numerical valued features (different from the version used in Bernadó-Mansilla & Garrell-Guiu 2003). We can indirectly conclude that MSWV may outperform XCS and UCS on a subgroup of datasets. In addition to that, MSWV does not exhibit the deterioration in performance reported in (Bernadó-Mansilla & Garrell-Guiu 2003) on datasets with high number of classes like *and*, *soy* or *vow* (see Table 9 for IDs and Table 10 for MSWV's results).

As shown in Table 10, MSWV is outperformed by all other six classifier systems on one single dataset, *ADL* that happens to be the dataset with the highest number of instances. As will be shown in section 3.3.5 the MSWV performance on this dataset is coincidental and not related to its size.

MSWV has outperformed all other classifiers on two datasets, *gls* and *jst* that have continuous numerical features and more than two classes. This may suggest that either MSWV does particularly well on this kind of datasets or, alternatively, the performance of the other six classifiers on datasets with continuous numerical features does not match their performance

on datasets with integer or nominal features (this second alternative may be particularly true for the SMO classifier).

Table 10: Accuracy rates (%) of the classifiers used in the study. The row labelled W-L-T counts the wins-losses-ties of one-on-one comparison between MSWV and another classifier. The last two rows show the z-value and the confidence level of the Wilcoxon signed rank test.

	<i>MSWV</i>	<i>NB</i>		<i>IB1</i>		<i>IB3</i>		<i>C4.5</i>		<i>PART</i>		<i>SMO</i>
aud	76.86	72.61	•	75.00		78.41		77.26		79.42	○	80.75
bal	86.62	90.53	○	79.28	•	86.74		77.82	•	83.17	•	87.42
brw	96.91	96.07	•	95.44	•	96.61		95.01	•	94.69	•	96.75
bpa	67.39	54.87	•	62.20	•	62.49	•	65.83		65.25	•	58.00
cmc	52.99	49.72	•	42.49	•	45.11	•	52.73		50.17	•	49.82
cra	85.72	77.86	•	81.57	•	84.96	•	85.57		84.45	•	84.88
crg	73.59	75.16	○	71.88	•	72.21	•	71.25	•	70.54	•	75.15
glc	72.01	49.44	•	69.95	•	70.00	•	67.62	•	68.74	•	57.34
h-c	82.44	83.33	○	76.04	•	81.82		76.93	•	77.99	•	83.89
h-h	83.40	83.95		78.33	•	82.07	•	80.20	•	81.12	•	82.79
hep	84.26	83.81		81.35	•	80.84	•	79.23	•	79.74	•	85.68
ion	91.03	82.17	•	87.09	•	86.01	•	89.74	•	90.83		88.06
irs	94.80	95.53		95.40		95.20		94.73		94.20		96.20
lbr	84.39	93.51	○	84.21		92.81	○	78.77	•	77.89	•	92.98
pmi	74.61	75.76	○	70.62	•	73.87		74.49		73.46		76.80
pri	44.96	49.71	○	34.34	•	44.99		41.39	•	40.86	•	46.96
snr	80.53	67.69	•	86.15	○	83.75	○	73.61	•	77.40	•	76.49
soy	91.40	92.81	○	90.07	•	91.07		91.65		91.27		92.97
veh	67.55	44.68	•	69.59	○	70.21	○	72.28	○	72.21	○	74.08
vow	87.47	62.90	•	99.05	○	96.99	○	80.20	•	77.67	•	70.59
vot	95.08	90.02	•	92.44	•	93.08	•	96.57	○	95.98	○	95.77
wne	96.01	97.47	○	95.11		95.84		93.20	•	92.08	•	98.76
yst	58.76	57.99	•	52.61	•	55.09	•	56.61	•	54.81	•	57.10
zoo	93.86	94.95		96.53	○	95.54		92.57		93.37		96.04
ADL	75.91	83.38	○	78.75	○	81.74	○	85.92	○	84.80	○	84.98
HTH	93.52	95.63	○	90.66	•	92.64		99.20	○	99.20	○	93.83
KRK	94.60	84.61	•	87.56	•	92.07	•	99.01	○	98.22	○	94.23
LED	74.33	74.08		62.48	•	74.25		74.28		74.23		74.03
LTT	76.29	64.20	•	93.02	○	91.55	○	81.33	○	82.42	○	79.18
MUS	99.89	94.43	•	100.0		99.91		100.0		99.88		100.0
ODG	92.93	90.71	•	97.84	○	98.05	○	86.55	•	87.70	•	97.12
SAT	61.91	61.42		69.04	○	69.53	○	59.62		61.35		69.63

SEG	92.98	79.22	•	95.52	○	94.35	93.50	93.90	90.71
SIC	97.10	93.04	•	95.30	•	95.94	98.25	○	97.89
SPL	95.63	94.54		73.01	•	77.39	•	92.05	•
WVF	82.27	79.75		71.29	•	75.88	•	73.45	•
W-L-T		18-10-8		22-9-5		13-8-15		17-7-12	
z-value		2.31		2.2		0.31		2.26	
Confid.		97.91		97.22		24.34		97.62	
								97.62	
									-54.67

To verify whether the MSWV does indeed perform better than all the other classification algorithms on datasets with continuous numerical features we have identified 17 datasets that contain at least one such feature and performed a comparison between the 7 algorithms on this reduced set, using the same methodology. The results are shown in Table 11.

Table 11: Accuracy rates of the classifiers (in %) on datasets with at least one continuous numerical feature.

<i>Id</i>	<i>MSMV</i>	<i>NB</i>	<i>IB1</i>	<i>IBk</i>	<i>C4.5</i>	<i>PART</i>	<i>SMO</i>
bpa	67.39	54.87	•	62.20	•	62.49	•
cra	85.72	77.86	•	81.57	•	84.96	•
gls	72.01	49.44	•	69.95	•	70.00	•
h-c	82.44	83.33	○	76.04	•	81.82	•
h-h	83.40	83.95		78.33	•	82.07	•
hep	84.26	83.81		81.35	•	80.84	•
ion	91.03	82.17	•	87.09	•	86.01	•
irs	94.80	95.53		95.40		95.20	
lbr	84.39	93.51	○	84.21		92.81	○
pmi	74.61	75.76	○	70.62	•	73.87	
snr	80.53	67.69	•	86.15	○	83.75	○
vow	87.47	62.90	•	99.05	○	96.99	○
wne	96.01	97.47	○	95.11		95.84	
yst	59.00	57.99	•	52.61	•	55.09	•
HTH	93.52	95.63	○	90.66	•	92.64	
SEG	92.98	79.22	•	95.52	○	94.35	93.50
SIC	97.10	93.04	•	95.30	•	95.94	○
W-L-T		9-5-3		11-3-3		7-4-6	10-1-6
z		1.62		1.76		1.01	2.47
Wilcoxon		89.48		92.16		68.75	98.65
							98.98
							81.98

The results clearly show that MSWV outperforms all the other classifiers on the group of 17 datasets with at least one continuous numerical feature. On the subset (almost half of the original set was used for comparison), MSWV outperforms SMO with over 80% confidence and it outperforms both C4.5 and PART with close to 99% confidence.

The performance of MSWV on two large datasets that have noise artificially added, *LED* and *WVF*, is also good, as shown in Table 10.

3.3.4 The Importance of Speciation

In section 3.3.2.3 we described how the automatic speciation process helps the decision rules discovery. In this section we are going to further investigate this mechanism using some examples. To show the implications and the importance of this second level of speciation we selected two datasets, the Mushroom dataset (*MUS*) and the Waveform dataset (*WVF*). Figure 11 shows the fitness average as well as the number of subspecies of the two species/classifiers of the *MUS* dataset.

Figure 11: The fitness average (times 0.1) and the number of subspecies for the *MUS* dataset with speciation turned on. Testing accuracy = 99.89%.

Initially each individual is treated as a separate subspecies and consequently each of the two species starts with 200 subspecies (see equation (3.5) and Table 9 for *MUS* number of instances). Through repeated speciation, the number of sub-species decreases significantly until in the case of species 0 all the individuals end up in one subspecies. Paralleling this process we can see the fitness average of the individuals in each species converging towards the maximum fitness threshold. It may also be noticed that following speciation there is also a small increase in fitness.

The significant decrease in the number of species suggests that few rules are needed for classification and that by turning speciation off one may see a longer convergence time and a decrease in the training and testing accuracy. This assumption is confirmed by the results of a run with speciation turned off as shown in Figure 12.

Figure 12: The fitness average and the number of subspecies for the two species of the *MUS* dataset, speciation turned off. Testing acc. = 96.80%.

While the number of subspecies in each species stays constant at 200 we see a decrease in the final fitness average of each species and an increase in the length of the run from 300 to 500 generations. The decrease in training accuracy (from 99.92% to 96.64%) is complemented by a decrease in testing accuracy from 99.89% to 96.80%.

However there are cases where the number of initial subspecies barely decreases through speciation, as in the case of the *WVF* dataset, suggesting that these datasets are ruled by a larger number of weak and strong rules. In Figure 13 we show the number of subspecies of the 3 species (*WVF* has 3 classes) barely decreasing from the initial value of 200.

Figure 13: The fitness average and the number of subspecies for the *WVF* dataset with speciation turned on. Testing accuracy = 82.27%.

In such cases one would assume that by placing from the beginning all the individuals of a species in one single population (subspecies) and letting them compete freely we should see some individuals with a higher fitness emerge and also a faster convergence time. The accompanying decrease in diversity should result in many classification rules being lost and result in a decrease in overall training and testing accuracy (in spite of evolving individuals with a higher fitness!).

The results of a second run on the *WVF* dataset with all three species initialized with all individuals placed together in one population confirm all these assumptions (see Figure 14).

The length of the run decreases from 1220 generations to 340 generations (almost a factor of 3) and the fitness average of the population in each species increases! However, these positive

factors are complemented by a severe decrease in testing accuracy from 82.27% to 70.90%, a decrease of more than 10% (the training accuracy decreases from 85.36% to 76.54%, also a decrease of almost 10%).

Figure 14: The fitness average and the number of subspecies for the *WVF* dataset with all individuals placed in 1 subspecies. Testing accuracy 70.90%.

In order to be able to statistically estimate the contribution of the automatic speciation we have extended the experiments described in this section to the first 10 datasets in Table 9, for a total of 12 datasets as seen in Table 12.

On each dataset we have performed two experiments. In the first experiment we place each individual in its own subspecies (just like in MSWV) but we turned the speciation off ($T_s = \infty$). In the second experiment we placed all the individuals of a species in one subspecies ($M_s = 1$) and let them freely compete with each other. The results of ten ten-fold validation runs (with the results in the 'ON' column duplicated from Table 10) are presented in Table 12.

Following the methodology described in section IV a result is marked with $\circ$ if it shows a significant improvement (over 99% confidence level) over the normal case (shown in column

'ON'), and with ● if it shows a significant degradation. The last two rows show the z-value and the confidence level of the Wilcoxon signed rank test.

Table 12: Accuracy rates (%) of MSWV with speciation turned on, turned off and with all individuals of a species placed initially in one subspecies.

ID	ON	OFF	1 SS
aud	76.86	80.18°	75.62
bal	86.62	85.17°	70.18°
brw	96.91	96.94	95.24°
bpa	67.39	66.70	62.87°
cmc	52.99	50.13°	53.10
cra	85.72	85.35°	85.26
crg	73.59	70.02°	69.85°
gls	72.01	72.29	65.93°
h-c	82.44	82.05	76.53°
h-h	83.40	81.02°	80.48°
MUS	99.89	96.80°	99.68
WVF	82.27	81.94	70.90°
z-value		1.94	2.96
Confidence		94.76	99.69

Based on the results shown in Table 12 we can tell that the automatic speciation method used by MSWV improves with a confidence close to 95% the case when speciation is turned off and it improves with over 99% confidence the case where all the individuals are placed in one big population and allowed to compete freely (in such a case the speciation mechanism doesn't work because all the individuals are forced to belong to the same subspecies).

Although the automatic speciation works in the general case there are certain datasets for which placing all the individuals in one large population (subspecies) results in a better overall average. As shown in Table 12 such a dataset is *cmc*. In the case of the *and* dataset we obtained a significantly better average by turning speciation off. This result may be explained by the fact that *and* has many classes with a very small number of examples (1,2,3,4, etc.) forcing the lower population limit in equation (3.5). When speciation is turned on, such species will very often end up with 1 subspecies and the mistakes made on the negative examples are not compensated through statistical means (as they are when the speciation is turned off and all the individuals get to vote on such instances during testing).

3.3.5 Speed and Scalability

In this section, we discuss the speed and scalability performance of the MSWV system. In Table 13 we show the training time of the 7 algorithms on 10 large datasets.

Table 13: Training time (in seconds) on 10 large datasets.

Dataset	MSWV	NB	IB1	IB3	C4.5	PART	SMO
ADL	8630.6	0.5	0.05	0.1	8.9	148.3	2960.0
HTH	1537.0	0.03	0.01	0.01	0.27	0.36	93.8
KRK	1364.2	0.02	0.02	0.02	0.2	0.3	2.8
LED	1580.5	0.02	0.02	0.01	0.3	0.2	27.0
MUS	909.3	0.02	0.01	0.03	0.05	0.2	8.6
ODG	20864	0.45	0.01	0.33	2.47	6.65	23.1
SEG	1065.1	0.11	0.01	0.05	0.6	0.77	8.52
SIC	1152	0.02	0.02	0.02	0.44	0.44	0.98
SPL	3727.6	0.03	0.05	0.02	0.66	1.39	153.0
WVF	3644.4	0.22	0.05	0.01	2.58	12.1	2.23

In Table 14 we show the testing time of the 7 algorithms on 10 large datasets.

Table 14: Testing time (in seconds) on 10 large datasets.

Dataset	<i>MSWV</i>	<i>NB</i>	<i>IB1</i>	<i>IB3</i>	<i>C4.5</i>	<i>PART</i>	<i>SMO</i>
ADL	45.3	1.2	2035.0	3633.6	1.1	3.7	1.7
HTH	2.2	0.27	19.55	33.17	0.05	0.03	0.11
KRK	2.5	0.05	15.10	22.63	0.03	0.03	0.06
LED	1.8	0.11	11.16	20.56	0.09	0.09	0.22
MUS	0.4	0.14	62.45	106.25	0.06	0.06	0.44
ODG	39.7	9.33	170.54	217.0	0.16	0.11	1.1
SEG	3.1	1.26	8.96	12.85	0.11	0.01	0.05
SIC	1.1	0.14	20.5	32.7	0.03	0.03	0.11
SPL	2.0	0.19	23.7	36.5	0.03	0.05	5.5
WVF	10.7	2.14	98.4	129.0	0.11	0.11	0.16

1. The results shown in Table 13 indicate that the training in MSWV is many orders of magnitude slower than the other classifiers on the majority of the datasets.
2. For the Instance-Based classifiers the **testing phase** is the slow phase so the previous statement would consider the times in Table 14. In fact, IB3 testing phase on ADL dataset is slower than the SMO training time!
3. The MSWV speed is relatively constant when the number of training instances, number of attributes and classes are considered. If the training time is converted to milliseconds and divided by the product of these three parameters of a dataset (we call this number: **normalized training time**) we get the results shown in Table 15. It may be seen that the normalized training time for MSWV does not vary more than 2.56 (max/min) on this set of datasets. On the other hand, the same ratio for SMO is 59.

This suggests that using the average time of 14.84 [ms] one can predict the training time for a given dataset with acceptable accuracy.

4. The same results suggest that SMO's running time depends nonlinearly on the data type and dataset parameters while MSWV may in fact have a linear dependency.

Table 15: Normalized training time for MSWV and SMO in [ms]

Dataset	<i>MSWV</i>	<i>SMO</i>
ADL	18.9	6.493
HTH	10.5	0.643
KRK	18.8	0.039
LED	11.3	0.193
MUS	7.6	0.072
ODG	17.4	0.019
SEG	18.2	0.083
SIC	10.4	0.013
SPL	15.8	0.799
WVF	19.5	0.011
Mean	14.84	0.836

5. To study the dependency of the training time with the number of samples in the dataset, we have selected the ADULT dataset and, using the same holdout estimate methodology for training and testing we performed 4 more additional runs with 1/2, 1/4, 1/8 and 1/16 of the original number of samples. The results presented in Table 16 show clearly that while the MSWV training time decreases roughly by a factor of 2 when the number of instances is halved, the SMO training time decreases by a factor of 4! This seems to suggest that while the MSWV training time increases as $O(N)$ the

SMO training time increases as $O(N^2)$. This finding suggests that MSWV scales better than SMO.

6. A similar analysis on PART, IB1 and IB3 showed PART to increase both the training and testing time as $O(N^2)$ while IB1 and IB3 increase their test time as $O(N^2)$. C4.5 and NB seem to increase their training time as $O(N)$ just like MSWV, but with a smaller scaling constant.
7. Another finding coming off Table 16 is that while the accuracy of MSWV has increased for ADL/16, it has generally stayed around 75% for the other ADL fractions proving that its performance on the ADULT dataset is not related to its size but it is rather coincidental.
8. MSWV's testing time, although much smaller than the training time also increases linearly with the number of instances in the dataset. The larger ADL sizes seem to indicate that SMO testing time increases more than twice when the dataset size increases twice.

Table 16: Training time and accuracy for fractions of ADL dataset.

Dataset	Size	MSWV			SMO		
		Accuracy	Train [s]	Test [s]	Accuracy	Train [s]	Test [s]
ADL	48842	75.91	8630.6	45.3	84.98	2960.0	1.7
ADL/2	24421	75.50	4328.7	24.6	84.28	557.3	0.6
ADL/4	12210	74.53	2438.5	10.7	83.93	117.5	0.30
ADL/8	6105	74.79	1252.0	7.6	83.09	18.7	0.14
ADL/16	3051	77.13	793.4	4.8	82.44	4.0	0.08

The conclusions of our analysis are:

- MSWV's training time may be roughly estimated using the approximate formula $T = 14.84 * n * N * L$ (see variable conventions in section 3.3.1).
- MSWV is much slower than the other algorithms when tested on small size datasets
- MSWV's scales better than SMO, IBk and PART and scales equivalently to NB and C4.5.

3.3.6 Final Comments about MSWV

MSWV uses a GA to evolve populations of decision trees (each population a 2-class classifier) but uses a calculated population size and a training stopping criterion rather than some fixed values. Other variables (like speciation period or fitness averaging interval) have been kept constant in all our experiments supporting the claim that MSWV is a relatively parameter free GBML system. An innovative speciation mechanism ensures that strong and weak rules are pursued by the evolving population, at the best evolution pace made possible by the initial random distribution of the population on the classification rules' landscape. The restriction that the decision tree be constructed on attributes built on distinct features clearly helps the search but it seems to introduce a limit on the complexity of the decision rules that can emerge in such trees and to negatively affect the classifier accuracy. This potential problem is successfully compensated by the statistical power of a voting system, making classification a collective task. Tested on thirty-six real-world learning tasks MSWV significantly outperforms (with over 97% confidence) a number of well-known classification algorithms and it is outperformed by SMO (with 55% confidence). When the comparison is done on a subset of 17 datasets with at least one continuous numerical feature MSWV outperforms all the classification algorithms it was compared to. MSWV is 3 to 4 orders of magnitude slower than the other algorithms when compared on datasets with less than 1000 instances but it scales as $O(N)$ while SMO scales as $O(N^2)$. MSWV performs well on noisy datasets, can handle records with unknown values and does not require post pruning. Based on the experimental

results obtained on the UCI datasets we believe that, at this moment, MSWV is the best GBML system reported in the EC literature.

3.4 The CH as an EC Model

In Chapter 2 we have reviewed the state of the art literature and established some links to the concept of Computational Habitats, and, in this chapter, we have reviewed our work along the road from GA to the CH concept. The areas of research discussed in these two chapters and their relationship to Computational Habitats is informally illustrated in Figure 15.

Figure 15: Conceptual road from GAs to CHs.

In Table 1 we gave a summary of known issues with the classical GAs and in Figure 15 with blue and red we show areas of research that over time have emerged to deal with some of those problems. With red we tried to differentiate those areas of research that we have also contributed to.

There is really no timeline attached to Figure 15. The concepts have been placed along the axis such to suggest our subjective view on how close they are to the classical model of GA or to

the CH model. Therefore with yellow we show those areas of research that have been in large described in Chapter 2 and that we see as being closer to the CH concept, while the blue and red blocks are seen as closer to the classical GA concept.

Our goal, as stated in the introduction, is to create an environment that can be used for **modeling** and in the same time can be used for **computations and optimization problem**. This goal is not rooted in the ambition of finding the "holy grail" of EC but it is rather the result of a practical approach, perhaps a reflection of our industrial experience. If we can find an environment that can be successfully used for both major areas of practical applications of ECs, we may expect a faster progress and a reciprocal benefit as progress in one may immediately benefit the other.

In support for the CH concept, many of the problems summarized in Table 1 may find a resolution in the CH model as suggested in Table 17.

Table 17: Summary of issues and possible solutions in the CH model.

<i>Issue</i>	<i>Description of possible solution</i>
Solution complexity	Use problem decomposition methods followed by path of solutions of increasing complexity.
Genotype-phenotype	Uses genotype and phenotypic representation for agents as well as an artificial embryogenesis (development) process.
Genotype encoding	Variable size. Stochastic agent representation with intra and extra agent signaling. Development process facilitated by representation.
Genetic operators	Selectable from set. Open to development of new operators. Object of possible studies of emergence.
Selection methods	Natural selection. Diversity preserved through model stochasticity.
Speciation	CH model allows for new species to emerge, evolve and coexist.
Cooperation	Inherent to the model. Made possible by inter agent signaling.
Learning	Learning allowed by the model, directly linked to model's inherent phenotypic plasticity
Hierarchical solutions	Natural to computational habitats evolution.

If the CH concept seems to naturally answer the issues associated with classical GAs, how would the CH model look like?

To address this issue we looked at nature. If one of our goals is to use CHs as models of nature how can one avoid an insurmountable complexity? In nature, the issue of diversity is automatically taken care of by the (theoretically) infinite complexity of nature. In nature everything happens with a probability! There is no guarantee the best individual of a colony of organisms would reproduce. In nature this probability is always less than 1. A second insight is obtained by associating various components of a model with parts of a mechanism. The chances of various parts "fitting" together are higher if the "edges" of these parts are not rigid but rather "soft". In other words we would like to have a certain tolerance for imprecision. These two insights have strongly suggested **stochasticity** as the first fundamental characteristics of agents populating the Computational Habitat; therefore we call them stochastic agents.

The second fundamental characteristic of the agents is their **signaling** capability. Giving agents' signaling ability has been a much easier decision to take. Signaling is a very important characteristic of life, as suggested by examples from nature. At a low level there is intra and extra cellular signaling while at a higher level there is signaling between organisms sharing a natural habitat. In nature four out of five mechanisms for cell differentiation (a basic mechanism in development) are based on signaling (Stanley & Miikkulainen 2003). In addition to signaling between live organisms we can also talk about signaling between inert matter and an organism. Without a signaling ability we would miss important features that we want to have in Computational Habitats, like autonomy.

We have mentioned that agents in CHs will have a genotype and a phenotype but instead of a fixed mapping between the genotype and phenotype we want the phenotypes to undergo a full development phase, an **artificial embryogenesis**, the third characteristic of CH agents.

In fact this last characteristic fits well with the previous two. Canalization is described as "the tendency for development of a specific genotype to follow the same trajectory under different

developmental conditions (different environments or different genetic backgrounds)” (Hallgrímsson et al. 2003) and **one of the mechanisms of canalization is stochasticity** (Stanley & Miikkulainen 2003). Also, **the signaling capability facilitates implementing a complex genotype to phenotype development process** (Stanley & Miikkulainen 2003). In their paper Stanley & Miikkulainen use the term **artificial embryogeny** instead of artificial embryogenesis (AE). A complex artificial embryogenesis process is motivated by the necessity of being able to create a very complex phenotype from a genotype which is many orders of magnitude smaller, because a much smaller genotype also means a much smaller search space to deal with.

If the atom has been considered for a long time to be the smallest particle of matter, a **stochastic process with signaling ability** is the smallest particle (or the **artificial atom**) of the CH model. Because the phenotype of the stochastic agent is composed of many such artificial atoms running concurrently, signaling becomes a characteristic of agent’s internal and external functionality. Moreover, as we will see in the next chapter, the natural choice of notations for describing the functionality of stochastic agents will be an extension to the π -calculus, a calculus for concurrent communicating processes (Milner 1990, Milner et. al. 1992).

Last, just as a computational component was added to self-replicating loops or agents in the Avida model, we have added a **computational component** to stochastic agents to create the **computational agents**.

A population of stochastic agents and a given landscape (for example, a 2D cellular space), will create a **Stochastic Habitat**. A population of computational agents, a given landscape and a mathematical objective will create a **Computational Habitat**.

The next chapter will place, stochastic and computational agents and habitats on a theoretical foundation while the following chapters will describe the first implementation and results of Stochastic and Computational Habitats.

4 COMPUTATIONAL HABITATS – THEORETICAL FRAMEWORK

"Mathematical ideas originate in empirics, although the genealogy is sometimes long and obscure. But, once they are so conceived, the subject begins to live a peculiar life of its own and is better compared to a creative one, governed by almost entirely aesthetical motivations, than to anything else and, in particular, to an empirical science." – John von Neumann

Definition 1 (Webster's dictionary) Habitat: the place or type of site where a plant or animal naturally or normally lives and grows.

Definition 2 (informal) Stochastic Habitat (SH): a habitat populated with stochastic agents.

Definition 3 (informal) Computational Habitat (CH): a community of concurrent stochastic agents carrying a computational component, where problems and solutions live, evolve and interact.

Starting with these informal definitions we have added stochasticity to a calculus designed to describe concurrent communicating processes to create a calculus for Stochastic Habitats and Computational Habitats. This is the subject of the rest of the chapter.

4.1 The *CH*-Calculus

The Computational Habitat (*CH*) calculus has its roots in π -calculus (Milner 1990, Milner et. al. 1992), the Chemical Abstract Machine (the CHAM - Berry & Boudol 1992) and the generative communication methodology (more specific the *tuple* spaces and the structured naming) of Linda language (Gelernter 1985, Carriero and Gelernter 1989). We might have called it stochastic π -calculus if not for its multiple roots. In this section we present the agents or process expressions and their interpretation as they will be used in the models we plan to build.

1. The *run-one* process $\sum_{i \in I} \left(\frac{P_i}{p_i} \right)$ where the index set I is finite and p_i is the probability

the *run-one* process will behave like the process P_i . We have $\sum_{i \in I} p_i = 1$. In π -calculus for example (where the same process is called the *summation* process), although not specifically specified, it is understood that $p_i = 1 / \text{card}(I)$. The same thing is true for Linda (Gelernter 1985) but there, instead of *summation* process is called "the *or-statement*". Following π -calculus we write 0 for the empty summation and call it the **NULL** agent. This agent does nothing and following the **CHAM** rules (Barry and Boudol 1992) we may simply evaporate it or following Linda we may simply use a "star-statement" that lives forever and recursively cleans such NULL processes when they occur.

2. The *run-all* process $A = \prod_{i \in I} \left(\frac{P_i}{p_i} \right)$ where the index set I is finite and p_i is the

probability that process P_i runs at any time t . This agent consists of $\text{card}(I)$ agents running in parallel with probability p_i . There is no mathematical relation between probabilities p_i but there is an evolutionary one because, associated with each agent (process) there is a cost and a benefit that must balance and workout for an organism to exist. In our model these probabilities will be evolved as part of the chromosome of the basic organism. In π -calculus the same process is called the *composition* process while in Linda is called the "*and-statement*". In both, $p_i = 1, \forall i \in I$.

3. A prefix form $\bar{y}x.P$, $y(x).P$ and $\tau.P$

a. ' $\bar{y}x.P$ ' is called an output prefix or a negative prefix. $\bar{y}$ may be thought as an output port of the agent which contains it. $\bar{y}x.P$ outputs x at port $\bar{y}$ and then behaves like P . In general x is a *tuple* $[v_1 \cdots v_n]$ where, following Linda, the parameters v_i are *actual* but *may also be formal*. They also can be names. The equivalent

syntax in Linda would be $[out(\bar{y} v_1 \dots v_n); P]$. If x is the empty *tuple* it may be omitted and we can write $\bar{y}.P$. If P is the *NULL* agent it may also be omitted and sending a signal through a named channel simply becomes $\bar{y}$.

Note: A special case of this agent is $\bar{y} * .P$ which sends agent P on channel y (the *self* channel) and behaves like the *NULL* agent. In *CH*-calculus all agents carry a *self* channel (port). An agent that has no other port but the *self* port is called the *signal* (or messenger) agent.

b. ' $y(x).P$ ' is called an input or a positive prefix. y may be thought as an input port of the agent which contains it. $y(x).P$ inputs an agent z at port y and then behaves like $P\{z/x\}$ (the substitution of the bound name x with z in P). In general x is a *tuple* $[v_1 \dots v_n]$ where, following Linda, the parameters v_i are *formal* but *may also be actual*. They also can be names. The equivalent syntax in Linda would be $[in(y v_1 \dots v_n); P]$.

c. ' $\tau.P$ ' is called a silent prefix. $\tau.P$ performs the silent action τ and then behaves like P .

4. A *restriction* $(x)P$ specifies that only components of P can use the ports x and $\bar{x}$. In the Linda spirit, we say that x and $\bar{x}$ are not visible outside the *tuple* spaces associated with P (in other words they are private). In biology terms this corresponds to intracellular signaling.

5. A *match* $[x = y]P$

This agent behaves like P if the names x and y are identical, otherwise it behaves like the *NULL* process.

6. The *up transducer* process $y(x)\Delta\bar{z}x = (\Delta) \mid \bar{z}x$ receives x on input port y and sends it out into the enclosing environment through an output port $\bar{z}$ and behaves as Δ . In

terms of *tuple* spaces hierarchy, this mechanism can be used to move signals up the *tuple* spaces hierarchy.

The down transducer process $y(x)\nabla\bar{z}x = (\nabla | \bar{z}x)$ receives x on input port y attached to an agent, places x within the agent through a port $\bar{z}$ and behaves as ∇ . This mechanism can be used to move down the *tuple* spaces hierarchy. To borrow again from biology, the up and down transducer processes are the equivalent of signal transduction, a process by which extra cellular inputs lead to intracellular messages that modulate cellular physiology (and the other way around). The transmembrane receptor proteins are embedded in the cell membrane and act as signaling transducers. They are the biological equivalent to the up and down transducer processes. The *CHAM* equivalent mechanism is called the *airlock*.

Associated with each (computational) agent there is a set of *tuple* spaces making what can be interpreted as the private communication space. However, there is a clear need for the agents to communicate with other agents, and the communication must occur outside of their own restricted communication space.

Figure 16: The role of membrane protein receptors in cell signaling
(The Biology Project – University of Arizona).

The up and down transducer processes facilitate the communication between agents. This method allows two possible solutions in practical implementations. When studying emergence mechanisms we are interested in autonomy of agents and the messages between agents will actually be implemented as messenger-agents. Moving in and out of an organism-agent will be done through a transducer (up or down) process. This is the approach taken in Chapter 5 and 6. In certain studies focused on the optimization (and computational) capability of the Computational Habitats it would probably be more effective to simply use *tuple* spaces, which is closer to the approach we used in Chapter 7. The best practical route will be determined based on the type of experiment.

The last rule from the π -calculus, the process definition (Milner et. al. 1992) is not absolutely necessary. In the CH model agents will be created (through various reproduction mechanisms) and destroyed dynamically. The process definition rule is appropriate for compiler implementations of languages based of π -calculus. That being said, the substitution rule will be used:

7. We will write $\{P(y_i / x_i)\}_{i=1, \dots, n}$ for the simultaneous substitution of y_i for all free occurrences of x_i in P with $i = 1, \dots, n$.

A process $\begin{pmatrix} P \\ 1.0 \end{pmatrix}$ can be simply written as P and P may be considered a non-stochastic process.

4.2 Formal Representation of Stochastic Agents

In this section we formally describe the stochastic and computational agents using the *CH*-calculus. In this section when we say that P is a stochastic process we mean $\begin{pmatrix} P \\ p \end{pmatrix}, 0 < p < 1$.

Definition 4: Given a stochastic process P , y is called a **signaling space** for P , if and only if $\bar{y}x.P$ or $y(x).P$.

Note 1: Following definition 3 in *CH*-calculus y is called a signaling space for P if y is a channel on which a process P may receive or send a signal.

Definition 5: A signaling space y is called an **internal signaling space** for a stochastic process P if and only if $(y)P$.

Note 2: Following definition 4 in the *CH*-calculus an internal signaling space is a restricted channel (i.e. only P has access to it).

Definition 6: y is called an **external signaling space** for a stochastic process P if and only if y is a signaling space for P and y is not an internal signaling space for P .

Definition 7: A signaling space y that is not an internal signaling space for any stochastic process P is called a **habitat (or environment) signaling space**.

Definition 8: Following the syntax of *CH*-calculus, a **stochastic agent** $A(t)$ is a set of communicating concurrent stochastic processes $(P_i)_{i \in I}$ with the following properties:

1. $\exists i \in I$ and a habitat signaling space y such that y is an external signaling space for P_i
2. For $\forall i \in I, \exists j \in I$ such that $(x)(\bar{x}u.P_i \mid x(t)P_j)$ and for $\forall P \notin (P_i)_{i \in I}$ x is neither an internal or an external signaling space for P (i.e. for any stochastic process in the set there is another one in the set with which the first one can communicate through an internal signal space – or private channel)
3. $A(t)$ is a run-all stochastic process represented in general as:

$$A(t) = (x_j)_{j \in J} \left(\prod_{i \in I} \left(\begin{matrix} P_i(t) \\ p_i(t) \end{matrix} \right) \right) \quad (4.1)$$

4. The internal signaling space of agent $A(t)$ is $ISS = (x_j)_{j \in J}$ (the set of all internal signaling spaces of the constituent stochastic processes).
5. The external signaling space of agent $A(t)$ is $XSS = (y_k)_{k \in K}$ (the set of all external signaling spaces of the constituent stochastic processes).

Note 1: Definition 8 is very important because while it gives a general definition of life in Stochastic Habitats it also suggests how the most primitive form of life in a SH looks like.

Note 2: The dependence on time t is intended to suggest that, in general, the processes and their probabilities of running may be dynamically modified (phenotype plasticity).

Note 3: The communication between a stochastic agent and the environment (including other stochastic agents) or between various internal stochastic processes is based on signaling.

Proposition 1: Any stochastic agent contains at least one transducer stochastic process.

True based on points 1 and 2 of Definition 8 (above) and CH-calculus (Definition 6).

Definition 9: If $A(t)$ is a stochastic agent given by equation (4.1), and a stochastic process $B(t) \in A(t)$ is a stochastic agent according to Definition 8 with the only exception that XSS instead of being the habitat signaling space of $A(t)$ is the internal signaling space (ISS) of $A(t)$, we say that $B(t)$ is a stochastic sub-agent (or simply a sub-agent) of $A(t)$.

Definition 10: A stochastic process that is neither a transducer stochastic process nor a stochastic sub-agent is called a **worker stochastic process**.

Proposition 2: The simplest life form in a Stochastic Habitat is a stochastic agent with 1 or more transducer stochastic processes and 0 or more worker processes.

The simplest form of life is a stochastic agent with no sub-agents and the proposition follows directly from Definitions 8, 9 and 10.

Note 4: Proposition 2 raises some very interesting questions the most important being that nothing in it suggests that the simplest form of life has direct replication capability without which we cannot really talk about a life form! This area of "emergence of life in Stochastic Habitats" will be subject to future research. This research may very well prove that the requirement for replication may not be fulfilled without at least 1 worker stochastic process, in which case Definition 8 (and as a result Proposition 2) may have to change.

Definition 11: The set of stochastic processes defining a stochastic agent may be expressed as a reunion of three sets of processes: a set T of transducer stochastic processes, a set W of worker stochastic processes and a set S of stochastic sub-agents, $(X_i)_{i \in I} = T \cup W \cup S$. While virtue of Proposition 2, $T \cup W$ are the processes comprising the simplest form of life in SH, if $S \neq \emptyset$, we are going to call this stochastic agent the **System sub-agent**.

Note 5: Definition 11 allows us to see a stochastic agent as being a reunion of stochastic sub-agents. This view will prevail for the rest of this chapter and also for the following chapters.

Note 6: Each stochastic agent has one System sub-agent.

Note 7: In the rest of this chapter and in the following chapters we are going to use the following sub-agents.

Reproductive sub-agent, responsible for agent's reproduction

Feeder sub-agent, responsible for feeding the agent

Apoptotic sub-agent, responsible for agent's death

Computational sub-agent, when added we obtain a computational agent

Observational sub-agent, used for monitoring in Computational Habitats.

All agents have: a set of stochastic sub-agents, a genotype and a phenotype. When the phenotype or a part of a phenotype is the formal representation of a problem or a solution to a problem the organism is a computational organism and the habitat becomes a Computational Habitat.

Example 1: A stochastic agent A composed of two sub-agents, a System sub-agent P_s and a

Reproductive sub-agent P_r , may simply be represented as: $A = \begin{pmatrix} P_s \\ P_r \end{pmatrix} \bullet \begin{pmatrix} P_r \\ P_s \end{pmatrix}$ or, alternatively,

$A = \begin{pmatrix} P_s \\ P_r \end{pmatrix} \mid \begin{pmatrix} P_r \\ P_s \end{pmatrix}$ (which is the representation used in π -calculus to suggest parallelism).

This representation makes the agents zero-dimensional (point form). These agents are immortal as they do not have an apoptotic component. In this model the System sub-agent may for example correspond to a cell's membrane (the shell) while the Reproductive sub-agent would be just what the name suggests. These kinds of agents are just pattern replicators.

The agents described in this example are the basis for the implementation and the results described in Chapter 5, section 5.1.

Definition 12: The **genotype** of a stochastic agent is a set G given by the equation:

$$G = (g_i)_{i \in I} \quad (4.2)$$

where g_i are called **genes** and are given by the equation

$$g_i = (tP_i, aP_i, p_i) \quad (4.3)$$

where tP_i , a template representing the process P_i , is an implementation dependent object, aP_i represents the arguments of the process P_i and p_i is the probability of running from equation (4.1).

Note 8: Please note that both the System and the Reproductive sub-agents in Example 1 are

run-all processes in which case they may be represented as: $P_s = \prod_{i \in I} \begin{pmatrix} P_{s_i} \\ p_{s_i} \end{pmatrix}$

and $P_r = \prod_{j \in J} \begin{pmatrix} P_{r_j} \\ p_{r_j} \end{pmatrix}$. The simple model genotype of such an agent would be represented by

$$G = (g_{s_i})_{i \in I} \cup (g_{r_j})_{j \in J}.$$

Definition 13: A gene mutation operator is a function m acting on the genes of a genotype such that: $g' = m(g)$.

Example 2: The simplest gene mutator acts only on the probability part of a gene such that:

$$m((tP, aP, p)) = (tP, aP, p') \quad \text{where } p' = \text{limiter}_{(0,1)}(p + \text{Gauss}(0, sd)), \quad p, p' \in [0, 1].$$

where $\text{limiter}_{(a,b)}(x) = \begin{cases} a & \text{if } x \leq a \\ b & \text{if } x \geq b \\ x & \text{otherwise} \end{cases}$. This simple gene mutator is used in the study presented

in Chapter 5, section 5.2.

Definition 14: Some or all agents may carry a sub-agent $\begin{pmatrix} P_o \\ 1.0 \end{pmatrix}$ called an **observational sub-agent**. This sub-agent would act as a probe and can be used for data collection purposes by an external observer.

Definition 15: Some or all agents may carry a sub-agent $\begin{pmatrix} P_a \\ 1.0 \end{pmatrix}$ called the **apoptotic sub-agent**. This sub-agent is responsible for triggering agent's death (apoptosis) and it is further divided in three-subcomponents: $P_a \triangleq \begin{pmatrix} \bar{y} \cdot P_{ai} \\ P_{ai} \end{pmatrix} \bullet \begin{pmatrix} P_{aa} \\ 1.0 \end{pmatrix} \bullet \begin{pmatrix} P_{at} \\ p_{at}(n) \end{pmatrix}$ where P_{ai} is an **apoptosis inhibitor** process which is activated based on the reception of a signal on channel $\bar{y}$ (from

another sub-agent, for example the feeding one, a sort of keep-living signal) and will simply act to **decrease** the probability $p_{at}(n)$, P_{aa} is an **apoptosis accelerator** process and will always increase the probability $p_{at}(n)$ and P_{at} is the **terminator** process (the great Reaper) responsible for the death of the agent, that will trigger with probability $p_{at}(n)$. In computational studies mass conserving is not an issue so the agent will simply be removed from the landscape. This apoptotic mechanism is similar to the "withdrawal of positive signals" one of the many mechanisms used in nature for cell apoptosis (Cain 2000).

Example 3: A very simple apoptotic mechanism is implemented and discussed in Chapter 5 section 5.3. The changes made to the definition are:

- Both the System and the Reproductive sub-agent have an apoptotic process in their structure. If the apoptotic process in the Reproductive sub-agent triggers, the sub-agent becomes non-functional (we implement this by nullifying the object). If System's apoptotic process triggers, the agent is killed. So the apoptotic sub-agent functionality is collapsed with the System sub-agent functionality.
- $P_{ai} = null$, $P_{aa} = null$ and as a consequence $\begin{pmatrix} P_a \\ 1.0 \end{pmatrix}$ becomes $\begin{pmatrix} P_{at} \\ p_{at}(n) \end{pmatrix}$
- $p_{at}(n) = cst.$ which means that during the life of an agent the apoptosis probability remains constant
- $(tP_{at}, aP_{at}, p_{at}) \in G$ so the apoptosis probability is treated (at least in section 5.3) as a mutable gene component.

Example 4: A more sophisticated apoptotic mechanism is implemented in Chapter 6. It practically relies on the following changes:

- All sub-agents have an apoptotic process in their structure $\begin{pmatrix} P_a \\ 1.0 \end{pmatrix}$ with the one in the System being responsible for agent's apoptosis.

Instead of implementing $\begin{pmatrix} \bar{y}.P_{ai} \\ p_{ai} \end{pmatrix}$ we implement $\begin{pmatrix} (((er \geq \text{quanta})=\text{true}).P_{ai}) \\ p_{ai} \end{pmatrix}$ which should

be read: "if the sub-agent energy reserve is greater than quanta execute P_{ai} ". This simplification replaces an apoptotic specific signal with an energy check.

- $P_{aa} = \{p_{at}^* = 1.1;\}$ so the apoptotic accelerator increases the apoptosis probability 1.1 times
- $P_{ai} = \{p_{at} / = 1.0891;\}$ so the apoptotic inhibitor decreases the apoptosis probability 1.0891
- The two cases above are combined such that in fact p_{at} always increases, faster by 1.1 or slower by 1.01
- The $(tP_{at}, aP_{at}, p_{at}) \notin G$ (so the p_{at} is not mutable anymore). In the studies discussed in Chapter 6 we have used $p_{at}(0) = 1.0E-7$

Example 5: In the last section of Chapter 6 the apoptotic mechanism becomes an inherent component of **every process component** of every sub-agent. As a result of this approach in addition to sub-agent failures we would also have process failures. This very low level of implementation of an apoptotic process allows us to introduce a repair mechanism, as a way of recovering the failed processes, and study its effect.

Definition 16: If $A(t)$ is an agent given by equation (4.1) let's define $\mathcal{P}(t) \triangleq \{P_i(t)\}_{i \in I}$ the set of all process components of $A(t)$ at time t . A process $[P_y = \text{null}]P_x \in \mathcal{P}(t)$ is called a

stem process if and only if before the execution of process P_x at time $t = t_0$, $P_y \notin \mathcal{P}(t)$, and after the execution of P_x at time $t = t_0$, $P_y \in \mathcal{P}(t)$. If $P_y \notin \mathcal{P}(t), \forall t \leq t_0$ we say that P_y was created by the stem process P_x as a result of a **development process**. If $P_y \in \mathcal{P}(t), \exists t < t_1$, $P_y \notin \mathcal{P}(t), \forall t_1 \leq t < t_0$ and $P_y \in \mathcal{P}(t), t = t_0 + 1$ we say that P_y was **recreated** by the stem process P_x at time t_0 as a result of a **repair process**. The development and repair actions may be represented as reduction rule **DEV** as follows:

$$DEV: \frac{[P_y(t) = null]P_x \quad P_y \notin \mathcal{P}(t)}{[P_y(t) = null]P_x \quad \mathcal{P}(t) \cup \{P_y\}} \quad (4.4)$$

Note 9: In plain words, stem processes are processes in charge of creating other processes when they do not exist. This process creation is performed first during the agent development process. Subsequent recreations may take place as part of the repair mechanism.

Note 10: We shouldn't forget the process $[P_y(t) = null]P_x$ is a stochastic process so equation (4.4) is performed only when the process is running.

Example 6: The development and repair actions performed through reduction equation (4.4) are used only in the last section of Chapter 6 when the effect of the repair mechanism on the life expectancy of the stochastic agents is studied.

Example 7: Except for the last section, the rest of Chapter 6 uses only a simplified version of the DEV action as shown by the following reduction rule.

$$DEV: \frac{[P_y(t) = null]P_x \quad P_y \notin \mathcal{P}(t)}{P_x = null \quad \mathcal{P}(t) \cup \{P_y\}} \quad (4.5)$$

In the simplified DEV reduction rule, a stem process dies (and is subsequently removed from the running list) after fulfilling its role of creating the process for which it is responsible.

Definition 17: If A is an agent given by equation (4.1) and R_s and R_r are the System and the Reproductive sub-agents, **birth** is an action given by the following inferences:

$$\mathbf{BIRTH} : \bar{a}y.P_s \mid a(y).P_r \mid b(x)\Delta\bar{c}x \rightarrow P_s \mid \bar{b} * m(A).P_r \mid b(x)\Delta\bar{c}x \rightarrow A \mid m(A) \quad (4.6)$$

where $m(A)$ is a mutated version of the agent A .

Note11: Using the note following rule 3.a in the *CH*-calculus description, $\bar{b} * m(A).P_r$ should be read as 'The Reproductive sub-agent sends a mutated version of agent A through a self-channel b '. The result of the last inference shows agent $m(A)$ released in the environment and running in parallel with agent A .

Note 12: For clarity we again have not shown the probabilities of running associated with the two sub-agents and the up transducer process.

Note 13: We show the up transducer process as running in parallel with the P_s sub-agent process. In reality it is part of it. We simply depicted it this way to avoid bringing into the equation more processes than strictly necessary. This approach will be also used in following equations.

Example 8: The general equation (4.6) is used as such in Chapter 5 and Chapter 6. The difference occurs with the Reproductive sub-agent P_r on how $m(A)$ is produced. While in Chapter 5 $m(A)$ is produced by cloning A and applying the gene mutation operator, in Chapter 6 $m(A)$ is produced after a sophisticated development process, starting using the genotype G and applying equation (4.4) repeatedly until all the phenotypic processes are developed.

In the case of Chapter 5 the first inference rule of equation (4.6) could be developed into the following simple cloning rule.

$$\begin{aligned} \bar{a}.y.P_s \mid (a(y)\nabla\bar{u}y \mid u(y).P_c \mid v(x)\Delta\bar{b}x)_r &\rightarrow P_s \mid (\nabla \mid \bar{u}y \mid u(y).P_c \mid v(x)\Delta\bar{b}x)_r \rightarrow \\ &\rightarrow P_s \mid (\nabla \mid P_c \mid v^*m(A) \mid v(x)\Delta\bar{b}x)_r \rightarrow P_s \mid \bar{b}^*m(A).P_r \end{aligned}$$

In the above equation P_c is the cloning process, which as part of the Reproductive sub-agent upon the receiving of the "begin cloning" signaling will clone A and mutate it to produce $m(A)$.

In Chapter 6 the first inference rule of equation (4.6) is developed into a more sophisticated model for offspring producing:

$$\begin{aligned} \bar{a}y.P_s \mid (a(y)\nabla\bar{u}y \mid u(y).P_{so} \mid v(x)\Delta\bar{b}x)_r &\rightarrow P_s \mid (\nabla \mid \bar{u}y \mid u(y).P_{so} \mid v(z).P_p \mid w(x)\Delta\bar{b}x)_r \rightarrow \\ &\rightarrow P_s \mid (\nabla \mid P_{so} \mid v(z).P_p \mid \bar{v}z.P_{m(A)} \mid w(x)\Delta\bar{b}x)_r \rightarrow P_s \mid (\nabla \mid P_{so} \mid P_p \mid \bar{w}^*m(A) \mid w(x)\Delta\bar{b}x)_r \rightarrow \\ &\rightarrow P_s \mid \bar{b}^*m(A).P_r \end{aligned}$$

where P_{so} is the process which starts the offspring by creating an offspring process $P_{m(A)}$ and seeds the new offspring with a mutated chromosome, and P_p is a packaging process which is triggered when it receives a message on channel v signaling the completion of the offspring development process. The major difference between this model and the simpler model lays in the offspring process $P_{m(A)}$ which is added to the Reproductive sub-agent running list and must be run till the offspring is fully developed, using parent resources to develop.

Definition 18: If A is an agent given by equation (4.1) and R_s and R_f are the System and the Feeder sub-agents, **FEED** is an action given by the following chain of inferences:

$$\mathbf{FEED}: \bar{f}^*Food \mid (f(x)\nabla\bar{g}x \mid P_s \mid g(y).P_f) \rightarrow (\nabla \mid \bar{g}^*Food \mid P_s \mid g(y).P_f) \rightarrow (\nabla \mid P_s \mid P_f) \quad (4.7)$$

P_f converts the food into energy units and sends them back to the other sub-agents as illustrated by the energy distribution action called **DISTR**:

$$\mathbf{DISTR}: e(x).P_s \mid e(x).P_r \mid \bar{e}^*EU.e(x).P_f \quad (4.8)$$

Note 14: Please note that once the Energy Units (EU) signals are output by the Feeder sub-agent P_f , they are fair game for **all** the sub-agents to use including the Feeder sub-agent itself. Because all these actions are probabilistically performed, if, for example, re-absorbing the energy units back into the Feeder would not serve the agent well, through evolution the probability associated with such an action could be lowered, or alternatively, the gene associated with the re-absorbing, $e(x)$, may be simply deleted from the chromosome, and that is exactly what could be seen in certain cases discussed in Chapter 6.

Definition 19 (mobility): If we are interested, for example, in studying how mobility as a behavior can emerge, we will associate a relative coordinate with a component process. In this

case an agent will be represented as: $A = \prod_{i \in I} \begin{pmatrix} P_i^{\delta_i} \\ p_i \end{pmatrix}$ where δ_i is the relative coordinate of

the sub-agent (process) P_i . One sub-agent will always be placed at relative coordinates zero (for example the System sub-agent) and the other sub-agents will be placed following the rule that no more than one coordinate may change from an already placed sub-agent by $+1$ or -1 .

Example 9: Reformulating the previous example in a 2-dimensional grid-like landscape we may have the following possible five agent shapes:

$$\begin{aligned} A_1 &= \begin{pmatrix} P_s^{(0,0)} \\ p_s \end{pmatrix} \mid \begin{pmatrix} P_r^{(0,0)} \\ p_r \end{pmatrix}, & A_2 &= \begin{pmatrix} P_s^{(0,0)} \\ p_s \end{pmatrix} \mid \begin{pmatrix} P_r^{(0,-1)} \\ p_r \end{pmatrix}, & A_3 &= \begin{pmatrix} P_s^{(0,0)} \\ p_s \end{pmatrix} \mid \begin{pmatrix} P_r^{(0,1)} \\ p_r \end{pmatrix}, \\ A_4 &= \begin{pmatrix} P_s^{(0,0)} \\ p_s \end{pmatrix} \mid \begin{pmatrix} P_r^{(-1,0)} \\ p_r \end{pmatrix}, & A_5 &= \begin{pmatrix} P_s^{(0,0)} \\ p_s \end{pmatrix} \mid \begin{pmatrix} P_r^{(1,0)} \\ p_r \end{pmatrix} \end{aligned}$$

In a 3-dimensional grid-like space the agents may thus have seven possible shapes.

One can imagine that an agent can be made to move around by incrementally adding (or subtracting) one unit from one coordinate of all the sub-agents. If agent A_3 makes 2 steps to the left and five steps down in n consecutive time units it becomes:

$A_3(n) = \left(\begin{matrix} P_s^{(-2,-5)} \\ p_s(n) \end{matrix} \right) \mid \left(\begin{matrix} P_r^{(-2,-4)} \\ p_r(n) \end{matrix} \right)$. The coordinates of sub-agent P_f represent the displacement

vector with respect to the original starting point and the original shape of the agent is not lost (it can be retrieved using the displacement vector). The way the two associated probabilities were written suggests they may vary over time.

4.3 Formal Representation of Computational Habitats

Definition 20 (formal): A Stochastic Habitat is formally defined as a system $H = (\mathcal{P}, \mathcal{L})$ where $\mathcal{P}$ is a population of stochastic agents defined as in the previous section and $\mathcal{L}$ is a given landscape.

Definition 21 (formal): A Computational Habitat is defined as a system $C = (\mathcal{P}, \mathcal{L}, O)$ where $\mathcal{P}$ is a population of stochastic agents with some (or all) of them having a computational sub-agent component, $\mathcal{L}$ is a given landscape and O is a mathematical objective.

The formal description of a computational agent clearly indicates the support for communication (signaling) but if computations are to be performed it doesn't say anything about what language will be used for performing them. The reason we didn't say anything about it is because the model is abstract and independent of what the choice for a processing language might be. Similar to how Linda implementations take the form of C-Linda or Lisp-Linda, we may very well have a Lisp-CH, C-CH or NN-CH where the later one would be a Computational Habitat where computations are performed by Neural Networks. The advantage of a Neural Network based implementation is a better capacity to learn.

The end of simulation is externally decided by using the information supplied through the observational sub-agents defined in the previous section.

4.4 More Examples

Example 10 (formal): Computational chromosome

This is an example that describes the computational chromosome of an agent involved in building a 32-bit string solution (to a bit string problem) and the process for building it. Therefore the chromosome must include instructions for making 0s and 1s and instructions for building the solution. The process for sending the solution is not described.

When a quantity of energy q is received from another process (e.g. the feeder) the total amount of energy Q of the computational process is updated. One of three channels $\{x1, x0, xS\}$ will be activated with probability p_1, p_0 or p_s with $p_1 + p_0 + p_s = 1$ if the required energy is available.

$$xq(q). \{Q = Q + q\} \mid \left(\frac{\text{if}(Q > q_1) \{Q - = q_1\} \bar{x1}}{p_1} \right) + \left(\frac{\text{if}(Q > q_2) \{Q - = q_0\} \bar{x0}}{p_0} \right) + \left(\frac{\text{if}(Q > q_s) \{Q - = q_s\} \bar{xS}}{p_s} \right)$$

Following the channel activation (by outputting a signal on the respective channel) Q is updated (decreased by q_i the energy required to run the agent triggered by the activation of the respective channel).

The channel activated will interact with a chromosome shown on the next line to create the appropriate process.

$$x1.(y1(z).\bar{z1}) \mid x0.(y0(z).\bar{z0}) \mid xBS.BuildSolution \text{ (probabilities to run are equal to 1.0)}$$

For example, if channel $x1$ was activated process $y1(z).\bar{z1}$ starts running. This is a process that will send a 1 on the output channel z received as a value over the input channel $y1$. By

convention, if n identical processes P are running, their run-one ensemble process is

$$\text{considered to be } \binom{P}{1/n} + \dots + \binom{P}{1/n}.$$

How would the process **BuildSolution** look like? Assuming, for example, that we are building a 32-bit string, **BuildSolution** may initially (a formal compression is possible but we prefer this extended form for clarity) look as follows:

$$\{Init\} \binom{\overline{y}nch0}{0.9969}.ch0(b0)\{F_1\} \mid \binom{\overline{y}nch1}{0.0001}.ch1(b1)\{F_2\} \mid \dots \mid \binom{\overline{y}nch31}{0.0001}.ch31(b31)\{F_{end}\}$$

n in $\overline{y}n$ may be a 0 or a 1 (in other words depending on whether bit 0 is to be a 0 or a 1 we may have $\overline{y}0ch0$ or $\overline{y}1ch0$). One passes $ch0$ to a process $y0(z).\overline{z}0$ that will return a 0 for what will become bit 0 of the evolved solution, while the other one passes $ch0$ to a process $y1(z).\overline{z}1$ that will return a 1 for what will become bit 0 (remember that we may have a pool of processes $y0(z).\overline{z}0$ and $y1(z).\overline{z}1$). The function **Init** is executed only once when the process **BuildSolution** is first started (imagine there is a another process called Init that does the job we want and starts BuildSolution afterwards) and does one thing, initializes the solution $solution = ()$

Function $F_i, i = 1, 31$ will do the following two actions:

- will set probabilities for all the output channels up to ch_{i-1} to 0 and set (for example) the probability of outputting on channel ch_i to $0.9969 + (i - 1) * 0.0001$
- will concatenate the bit received on ch_{i-1} to the head of list that we called **solution**

With the update procedure presented here, it is clear that $\overline{yn}ch31$ will be executed with probability 1 and, consequently, assuming there are enough bit producing processes, we are going to get the last bit in the string and get to function F_{end} which will do two things (just like *Init*):

- sets the probability for $\overline{yn}ch31$ to 0 (to prevent asking for another bit)
- sends the *solution* to another process that knows what to do with it

After the execution of F_{end} that instantiation of *BuildSolution* process becomes inert.

The system exhibits some interesting features.

First, it allows for degenerate (shorter) solutions. Obviously these solutions will not bring the expected rewards but may bring partial rewards.

Second, with some changes, for every bit i the possibility of a faulty bit can be introduced by

including both $\begin{pmatrix} \overline{y0}ch_i \\ p_i^0 \end{pmatrix}$ and $\begin{pmatrix} \overline{y1}ch_i \\ p_i^1 \end{pmatrix}$ where one probability is very large and the other one

very small. In fact both the faulty bits and the degenerate solutions may be seen as expressions of phenotypic plasticity and allow for studies of Baldwin effect (Mitchell 2001). We may want to allow for faulty bits with a higher probability than having degenerate solutions, as this characteristic may have a larger payback and the effect may be easier studied..

Third, if we see the requests for various bits as the expression of certain genes, the system definitely permits learning by using the gene expression mechanism.

Example 11 (informal): Problem Decomposition and Trading Species

Let's assume the problem under study is a Royal Road function with fitness function $R_1(x)$ of a bit string (Mitchell 2001) given by: $R_1(x) = \sum_i c_i \delta_i(x)$, where $\delta_i(x) = \begin{cases} 1 & \text{if } x \in s_i \\ 0 & \text{otherwise} \end{cases}$ where s_i are given schemas and δ_i are coefficients associated with them. The case we study is an optimal 64-bit string broken up into eight (equal size) building blocks ($c_i = 8, i = 1, \dots, 8$).

This problem is usually studied using populations of 64-bit strings. One of the problems associated with a classical GA searching for a solution would be hitchhiking, whereby "bad" bits may be carried (and spread in the population) along some "good" schemas (Mitchell 2001).

Using a CH model, besides a complete problem species that would receive complete solutions in the 64-tuple space, we could have many decomposed problem species receiving their input in the n -tuple spaces with $n < 64$ and corresponding solutions species.

So let's assume we have two solution agents delivering their solutions in the 4-tuple space. The solutions delivered in the 4-tuple space could be read-in by two different agents of a problem decomposition 4-tuple species OR they can be read-in by an *intermediary* agent, a member of a trader-like species that will return some reward to the previous two 4-tuple solution agents and send a concatenated solution to the 8-tuple space expecting a bigger reward.

Previously we have mentioned that parameter matching is intended to follow the Linda paradigm that allows a special kind of *structured naming* that is similar to the "select" operation in relational database systems. For example a trader-like agent it may accept only solutions that have a certain pattern as some of the arguments. For example may accept only solutions of the kind $\{1, y, 1, z\}$, i.e. solutions that have values of 1 as the first and third parameter. Very interesting complex solutions may be built up as result of these convenient mechanisms built into CH model. This kind of collaboration between species can very quickly build up into a full optimal solution, especially to problems like this one where the solution exhibits symmetry.

Example 12 (informal): Evolving CAs to solve the density problem

In the case of the $\rho_c = 0.5$ density task, the problem agents may carry the initial cellular configurations and the solutions would represent the CA rules. The problem has been extensively studied in the context of CAs (Mitchell et al. 1994) and NUCAs (Sipper 1996) as mentioned in Chapter 2.

One problem of the classical GA studied by Mitchell et al., was the difficulty level of the test cases that was not adequately increasing as the rules found were getting better at solving the density task. A CH model would offer the advantage that both the solutions and the problem species would evolve and try to get "better" without the need for continuous external monitoring of the quality of the solutions and methods for increasing the difficulty of the problems not too early and not too late!

Because we are seeking a solution in the framework of CAs we wouldn't be able to take advantage of the findings of the work with NUCAs where it was proven that combinations of rules may lead to better solutions in certain cases ($r = 1$). However, we can still take advantage of problem decomposer lower order initial conditions that can be quickly solved by sets of symmetrical (non-optimal) rules. Mitchell et al. pointed out in their research there are reasons to believe the optimal solution set may be symmetrical (and indeed, the best known so far, the Gacs-Kurdyumov-Levin CA rule set is symmetrical). By very quickly finding some sets of symmetrical CA rules as solutions to lower order problems it may be easier to find a "network-of-viable-solutions" evolving towards an optimal solution (the best solution to the density problem, called the ABK rule, has been reported by Andre et. al. 1996 using genetic Programming) as the order increases.

5 HABITAT MODEL

"Few things are harder to put up with than the annoyance of a good example" –

Mark Twain

5.1 A Simple Model of Pattern Replicators

The first habitat implementation was designed in a toroidal 2D cellular environment with very simple agents that may reproduce by cloning themselves and may place the offspring in a randomly selected free cell in a Moore neighborhood of radius 1, if such a free cell exists.

When the mutation operator is not applied to the clones these agents are in reality pattern replicators and their theoretical model is described in Chapter 4 section 4.2 Example 1. The only resource available to the system, the cellular space itself, will get quickly exhausted and copies of patterns capable of replication will fill the space. In spite of their simplicity these agents have a stochastic nature, signaling capability as well as a genetic and phenotypic makeup. Figure 17 shows the structure of an agent and the cloning steps.

Figure 17: Stochastic agent model and cloning steps

The genes of real world organisms capture the blue print for various phenotypic structures and the processes that are used for building these structures. The complexity of these genes makes it impossible to simulate them with the computational power available nowadays. However, one can considerably simplify things by replacing the structures with their equivalent functionality, as described by a certain *method* object, and a probability number describing the probability that the *method* is performed. This probability can also be seen as an efficiency coefficient associated with the *method*. The reproduction process involves a number of signaling steps which are performed with certain probabilities emphasizing the stochastic character of the agents.

- Release "begin cloning" signal into the System sub-agent Internal Signal Space (SSA-ISS). This action is performed with probability p_1 .
- Using a transducer process, move the "begin cloning" signal from SSA-ISS to the Reproductive sub-agent Internal Signaling Space (RSA-ISS). This action is performed with probability p_2 .
- Receive the "begin cloning" signal with probability p_3 .
- Perform cloning and release the clone "as a special signal" into the RSA-ISS. This action is performed with probability p_4 .
- Move clone from RSA-ISS to SSA-ISS. This action is performed with probability p_5 .
- Move the clone from SSA-ISS into one of the free cells of SSA-XSS, a Moore neighborhood of radius 1, if such a free cell exists, otherwise destroy the clone. This action is performed with probability p_6 .

The randomly generated sextuple $(p_1, p_2, p_3, p_4, p_5, p_6)$ will determine the probability of a Stochastic Agent to successfully create and release clones into the cellular space. In

Figure 18 we show a simulation after 30 cycles. With red we represent the 20 agents randomly generated and distributed in the cellular space during initialization and with other colors we represent various generations of offsprings. Because programs tend to process in a linear, repetitive manner, to reinforce the stochastic nature of our agents, the signaling spaces are shuffled every cycle to ensure a random distribution of signals.

Figure 18: The agent distribution in the habitat after 30 cycles.

Legend: red- initial agents, blue- 1st gen. clones, green- 2nd, black – 3rd, turquoise - 4th and subsequent generations. Circle labeled 0 shows an agent with no offsprings and with label 1 points to an agent with 1st generation offsprings, etc.

The agents with a healthy (functional) communication system will successfully clone themselves and use all the available resources in the habitat at which point the reproduction slows down. After 100 cycles the landscape will look as in Figure 19 and we may recognize that the agent that didn't have an offspring by cycle 30 continues to have none while others have large colonies around them. Because from the 4th generation up all the agents are depicted with the same color we can see that in fact the cellular space gets filled up with new offspring

that surround the clusters of the first three generations of clones. Although 100 cycles have passed we also see one agent with a late 1st generation offspring which shows the power of this very simple habitat model of capturing characteristics specific to real-life models (such as having a large variance in the replication period).

Figure 19: The agent distribution in the habitat after 100 cycles

5.2 A Simple Model with Mutation

The experiments described in the previous section have been performed without using a mutation operator. In this section we describe the effect of a mutation operator that acts on the set of probabilities that constitutes the agent's chromosome. The mutation operator used for modifying the probability part of the genes of the clone offsprings is identical with the one used successfully in (Tulai & Oppacher 2002b) and is given by the equation:

$$p_i^{mut}(t) = p_i(t) + \text{Gauss}(0, sd_i), \quad p_i, p_i^{mut} \in [0, 1] \text{ where } sd_i, i = 1, \dots, 6 \text{ have been set to } 0.1$$

Does the mutation operator have any effect on the agents in the absence of an explicit exogenous fitness function and a strict 'à la GA' selection operator? To answer this question

we have studied the time it takes for the agents to fill in half of the cellular space which in our studies has been set to a 100x100 2D toroidal grid. The results of 10 runs with and without using the mutation operator have been summarized in Table 18.

Table 18: The time (in cycles) for the agents to fill in half of the cellular space, with and without mutation.

Without mutation			With mutation		
Min	Max	Mean	Min	Max	Mean
171	239	206.2	102	128	116.3

The remarkable thing about this experiment is that with mutation turned on the average time for the agents to fill in half of the cellular space is less than 60% of the length when mutation is turned off. This is clearly made possible by an endogenous fitness function and the result of the implicit selection rules governing a habitat.

Figure 20: Spatial distribution of agents at the end of simulation with mutation turned off.

Through mutation, agents better adapted to reproducing are produced and their number and efficiency in reproducing accelerates every cycle resulting in a dramatic decrease in the time

needed to consume the only resource available to the system which is the cellular space. Figure 20 and Figure 21 also show that at the end of the simulation, with and without mutation for $seed = 10$, the spatial distribution of the agents is somehow different. The placement of the agents in the cellular space is primarily affected by the distribution of the reproductively most active agents.

Figure 21: Spatial distribution of agents at the end of simulation with mutation turned on.

5.3 Doing Away with Immortality

In previous sections the agents were indestructible. In this section we are taking one more step towards the general structure of a stochastic agent by adding the apoptosis process and, associated with it, an evolvable probability of running the apoptosis terminator process p_{AT} was added to agent's chromosome. In the first experiment performed we started with 20 agents similar to the previous experiments. However, this time, by cycle 18 all agents but one have died! An investigation shows that for the agent still alive $p_{AT} = 0.001425846$.

Luck had it that by cycle 28 a clone was born giving some hope for life to go on for the lineage of this last agent which was confirmed over the next cycles with more agents being born

before life became extinct. The stoppage criterion of half of the cellular space being filled up was reached at cycle 768 as shown in Figure 22.

Figure 22: Spatial distribution of agents at the end of simulation with mutation turned on and an apoptosis component.

As expected, the average probability of dying for the population of agents at the end of the simulation was relatively small: 0.01181202. A more accurate value for this probability was obtained by changing the stoppage criterion to 80% of the cellular space and removing the last generation of agents from the average calculation. This run lasted 900 cycles and the value that we obtained for the average probability of executing apoptosis was: 0.00839176. A value of $p_{AT} = 0.0084$ has therefore been used in subsequent experiments.

5.4 Gene Exchange

Mutation was the only genetic operator used in previous experiments. In this section we present the results of experiments performed within the habitat of stochastic processes after introducing a gene exchange mechanism borrowed from nature.

Prokaryotes transfer genes in two general ways (Cain et. al. 2000). In **transformation**, bacterial individuals, called **recipients**, take up fragments of double-stranded DNA released into the environment by other bacteria, called **donors**. Donor bacteria release their DNA into the environment when they die from natural causes or in response to chemicals produced by other bacteria. The transformation mechanism evolved as a way of incorporating new genes into bacterial DNA. The second mechanism for gene transfer involves **plasmids** or **viruses**, called **vectors**. While being distributed to other bacteria these vectors may accidentally transfer some of the host's DNA.

Borrowing from nature we have introduced transformation as a mechanism for gene mixing between stochastic agents. When an apoptosis process triggers the death of an agent, fragments of DNA are left in the environment as part of a static signal message. When absorbed from the environment into the internal signaling space of an agent this signal has a chance of making it to the reproductive sub-agent where genes from the agent's chromosome get replaced by homologous genes in the fragment leading to gene mixing as shown in Figure 23. In our experiments we wanted to see if this mechanism for transferring genes may offer some advantages in creating faster reproducing agents. In Table 19 we show the results of 10 runs with the transformation mechanism turned on and off, respectively. In both cases the initial population was set to 100 agents randomly placed in a 100x100 2D toroidal space.

Table 19: The time (in cycles) for the agents to fill in 80% of the cellular space, with and without transformation.

Without transformation			With transformation		
Min	Max	Mean	Min	Max	Mean
140	201	164.7	139	208	169.1

Using the results in the previous section the probability of running of both apoptosis processes (for the System sub-agent and for the Reproductive sub-agent) were preset to 0.0084 for all the agents. The probability of running for all other processes was randomly set.

Figure 23: Gene mixing using the transformation mechanism borrowed from prokaryotes.

A paired two-sided t -test at 95% confidence level found the two sets of results equivalent (no significant difference) that we attribute to the static nature of the chromosome carrying signals.

One legitimate question is linked to the definition of "habitat signaling space" for an agent (see Definition 7 in Chapter 4) because this definition has a big impact on how the chromosome fragments spread in the environment once released. In the experiments presented in this section we have set the habitat SS to a Moore neighborhood of size 1 (i.e. 9 cells including the cell where the agent is located). In experiments the habitat SS is setup in a separate cellular space and while two agents cannot occupy the same grid cell, signals can.

5.5 The Feeder Sub-Agent

All experiments performed so far had two important characteristics:

1. There was no cost associated with any kind of processing
2. The only resource shared by the agents was the cellular space

Both characteristics are specific to an ideal, low complexity world that offers little to encourage competition or other mechanisms for increasing complexity and unbounded evolutionary capability (Bedau et. al. 1998). In this section we take additional steps towards such objectives by adding the following enhancements to the model:

- With each sub-agent we associate an energy reserve, similar to the agents in the Echo model (Holland 1992, Forrest & Jones 1994, Graham & Oppacher 2003)
- We associate a cost of 1 energy unit with the processing performed by each stochastic task (for example sending or receiving a signal)
- The structure of the stochastic agents is enhanced with a "Feeder" sub-agent that has the following characteristics:
 - It has identical general structures as any sub-agent (signaling-space, sending and receiving signals, etc.)
 - It receives and decodes "food" signals and uses their contents to increase the "acquired energy" counter
 - It distributes the contents of the "acquired energy" counter to replenish the "energy reserve" of all the sub-agents (in this case the System sub-agent, the Reproductive sub-agent and the Feeder sub-agent energy reserve). The percentage of how much of the "acquired energy" goes to each sub-agent is given by evolvable genes normalized by a 'Feeder development' process.
- The various energy counters are initialized as it follows:
 - Feeder acquired energy = random number in the interval (100.0, 200.0)
 - Feeder energy reserve = random number in the interval (20.0, 40.0)

- System energy reserve = random number in the interval (20.0, 40.0)
- Reproductive energy reserve = random number in the interval (20.0, 40.0)
- The energy counters of an offspring are initialized to the above values without a corresponding decrease of any of the parent's energy counters (i.e. it is done ideally without conservation of an energy balance).
- The probability of running all three apoptosis tasks was initially set at 0.0084 (using previously obtained results) but was allowed to evolve.
- The environment is "seeded" with food signals with energy contents given by a random number in the interval (100.0, 200.0). No more "food" signals are injected into the system from outside.
- However, when an agent dies, a food signal carrying a random energy contents in the interval (100.0, 200.0) is left behind at the location previously occupied by the agent.

With these significant enhancements to the model we have first studied the viability and the evolution of such a model when the input habitat signaling space was set to a Moore neighborhood of size 0 (which contains only 1 cell, the cell where the agent is located). Again the stopping criterion was set to the agents filling 80% of the cellular space and the number of initial agents was set to 100.

As shown in Figure 24, after 3000 cycles the system seemed to have been stabilizing at just over 70% fill up of the cellular space, a figure clearly linked to our choice for initial energy reserves as well as the energy contents of the food signals planted in the environment.

Figure 24: The evolution of population size with 100 initial agents.

However, if the number of initial agents is changed to 1000, the 80% threshold is attainable after 2252 cycles as can be seen in Figure 25.

Figure 25: The evolution of population size with 1000 initial agents.

At the end of the simulation (depicted in Figure 25) we have averaged the values of the alleles for the 23 genes making up the chromosome and the results are given in Table 20.

Table 20: The average values of the alleles calculated over the whole population at the end of simulation (except for the last three the rest of them are probabilities of running the corresponding process).

Sub-agent	Process	Value
System	txSignal	0.3924075
	rxSignal	0.4412014
	txSignalUpTransducer	0.3962256
	rxSignalUpTransducer	0.0420392
	runReproductiveAgent	0.6449553
	runFeederAgent	0.5905957
Reproductive	apoptosis	0.0076211
	txSignal	0.1609777
	rxSignal	0.6748887
	txSignalUpTransducer	0.6405551
	rxSignalUpTransducer	0.6443559
	produceClone	0.6718141
	apoptosis	0.0042829
Feeder	energyDistributionGenesis	0.6832901
	txSignal	0.3205543
	rxSignal	0.4423394
	txSignalUpTransducer	0.6557566
	rxSignalUpTransducer	0.403337
	energyDistribution	0.6479087
	apoptosis	0.0052357
	systemDistributor	0.3372791
	reproductiveDistributor	0.1655982
	feederDistributor	0.3124366

The previous two experiments show that, while the distribution of agents as well as their number remains random, such habitats can achieve a stable state without additional infusion of energy from outside of the system. This capability can be clearly linked to our choice for a Moore neighborhood of size 0 because the only food an agent can consume is the one offered by its own grid cell, and after an agent's death a random value food signal is placed back in the grid cell. It was clear that by using a Moore neighborhood of size 1 as a habitat signaling space the agents would be given a chance to consume the food resource in an area larger than their own cells (9 cells versus only 1 cell) while producing a single food source upon death. It would appear that such a system is not sustainable in the long run. To verify this we have performed another experiment using an initial population of 1000 agents and a Moore neighborhood of size 1. As shown in Figure 26, to our surprise the system proved to be viable and the 80% stopping criterion was reached after 9552 cycles. How was that possible?

Figure 26: The evolution of population size with 1000 initial agents and a Moore neighborhood of size 1.

The averaged values of the alleles calculated over the whole population at the end of the simulation depicted in Figure 26 are given in Table 21.

Table 21: The average values of the alleles calculated over the whole population at the end of simulation (corresponding to Figure 26).

Sub-agent	Process	Value
System	txSignal	0.0964447
	rxSignal	0.0998905
	txSignalUpTransducer	0.1100311
	rxSignalUpTransducer	0.0012222
	runReproductiveAgent	0.1993847
	runFeederAgent	0.1922804
	apoptosis	0.0028186
Reproductive	txSignal	0.1284435
	rxSignal	0.6793953
	txSignalUpTransducer	0.6300401
	rxSignalUpTransducer	0.6319355
	produceClone	0.6863952
	apoptosis	0.0025303
Feeder	energyDistributionGenesis	0.6374911
	txSignal	0.1743409
	rxSignal	0.1012754
	txSignalUpTransducer	0.6688533
	rxSignalUpTransducer	0.1871100
	energyDistribution	0.6657359
	apoptosis	0.0020649
	systemDistributor	0.1903468
	reproductiveDistributor	0.2525185
	feederDistributor	0.4071476

In the rest of the section we are going to make some comments based on the results shown in Table 20 and Table 21.

The first comment will also give the answer to how it was possible for the system using a Moore neighborhood of size 1 to be viable.

For the food units to be absorbed by the agents and converted into energy we need the following chain of tasks to be activated:

```
System.rxSignalUpTransducer  
Feeder.rxSignalUpTransducer  
Feeder.rxSignal  
Feeder.energyDistribution
```

"System.rxSignalUpTransducer" is the first step in the chain of converting external food into internal energy; the probability of executing this task is only 0.0420392 for the Moore neighborhood of size 0 and 0.0012222 (!) for the Moore neighborhood of size 1. These values can be explained only by the fact that the agents have enough initial energy to replicate without additional external energy (remember that offsprings are generated with exactly the same energy configuration as their parents without depleting their parents' energy reserves). However, external food does help to extend the functionality of an agent (and therefore their capability to reproduce) but because the food source is not replenished, it is enough to have a single food intake over the lifespan of an agent (i.e. to input one food signal). If the process "System.rxSignalUpTransducer" would have a much higher probability to execute it would just consume energy by doing that while not helping the system with inputting more food (because there is none left after the first food signal is read). The same logic applies to the experiment with a Moore neighborhood of size 1 but in this case the number of food messages is 9 instead of 1 and the simulation is a lot longer.

A second comment is related to the fact that the apoptosis probabilities continued to decrease (they were all initially set to 0.0084 but they were left to evolve) therefore extending the average life expectancy of the agents.

A third observation is that probabilities of the four processes involved in reproduction:

```
Reproductive.rxSignalUpTransducer,  
Reproductive.rxSignal,  
Reproductive.produceClone  
Reproductive.txSignalUpTransducer
```

are all higher than the expected value of 0.5 (the probabilities are initially set by a uniformly distributed random variable). The probability for a clone to be born and present in the internal signaling space of the System sub-agent is given by the product of probabilities for the previous 4 processes to run times the probability that a "Begin Cloning" signal is issued which happens every time System.txSignal runs. In the case of the results in Table 21 this product is about 0.0179 which is in fact almost 10 times lower than the probability of the process in charge of releasing the new born in the environment which is System.txSignalUpTransducer.

A last observation is that from all the System processes the most important in the functioning of the agent are System.runFeederAgent and System.runReproductiveAgent and this is reflected in the probabilities of these processes to run which are higher than the probabilities of other System processes.

5.6 Autopoietic Habitats

An autopoietic system is operationally closed with no apparent inputs and outputs (Varela et. al. 1974). The habitat introduced in the previous section cannot exactly qualify as an autopoietic system because when new offsprings are born they receive an initial energy without appropriately decreasing the energy reserves of the parents therefore energy is injected into the system from outside. In this section we show that life in an autopoietic habitat is sustainable in the case of a Moore neighborhood of size 0 but it is not sustainable for a Moore neighborhood of size 1.

The only changes that we performed to the system described in the previous section are:

1. The initial seeding of the environment which now is done with "food" signals with an energy content given by a uniform random number in the interval [3000.0, 4000.0].
2. For an offspring to be produced the energy reserve of the Reproductive sub-agent must be sufficient to cover the initial energy requirements of the newborn which, upon birth, are subtracted from the energy reserve of the Reproductive sub-agent.

With these changes it becomes clear that an agent cannot produce an offspring without first absorbing a food signal from the environment, converting it to energy and distributing enough energy to the Reproductive sub-agent to cover the energy requirements of an offspring.

The first experiment was performed with an initial population of 100 agents and the stopping criterion set at 80% of the cellular space fill up with agents. The system proved to be stable and it reached the stopping criterion after 18243 cycles. The evolution of the population size was captured after 18000 cycles as depicted in Figure 27.

Figure 27: The evolution of population size with 100 initial agents, a Moore neighborhood of size 0 and energy conservation.

The averaged values of the alleles calculated over the whole population at the end of the simulation depicted in Figure 27 are given in Table A.1 in the Appendix..

The experiments performed with a Moore neighborhood of size 1 failed when we either used 100 or 1000 initial agents with results after 7000 cycles shown in Figure 28 and Figure 29.

Figure 28: The evolution of population size with 100 initial agents

Figure 29: The evolution of population size with 1000 initial agents

The reason the experiment with an initial population of 100 showed a better population growth than the experiment using an initial population of 1000 agents is that in the latter case the available food in the environment was quickly consumed by agents that had a high probability for inputting food signals (i.e. executing `System.rxSignalUpTransducer`) but didn't have well tuned Feeder and reproductive sub-agents. In the case of the lower initial

population it was enough to have one such good agent that in time would create offsprings capable of reproducing when in contact with environment cells not depleted of food by initial poor quality agents. This experiment gives a clear indication that starting with a larger population does not always guarantee a higher survival rate for it! Generally speaking, the environment must be rich enough at the time the first functional agents emerge.

The only way to prevent a population of stochastic agents using a Moore neighborhood of size 1 to die was to use periodic flooding (Hutton 2002), a phenomenon whereby at regular intervals we "flood" the habitat with food. Using a "flooding" period of 5000 cycles we were able to achieve a stable system with capabilities of filling close to 80% of the habitat cells as shown in Figure 30. One can easily identify the timing when flooding occurs as it is always followed by a sharp increase in population. Another characteristic of the system towards the end of a simulation of over 70000 cycles is the smaller drop in population between flooding events accompanied by a slower population decrease process as indicated by the slope of the population curve going down, which indicates a gradual adaptation to feeding cycles.

Figure 30: The evolution of population size with 100 initial agents and flooding every 5000 cycles.

A few comments regarding the results in Table A.1 in the Appendix are warranted.

First, when compared to the results shown in the previous section it is remarkable how the importance of being able to quickly absorb food from the environment is reflected in the increase in the probability of running the process `System.rxSignalUpTransducer` as well as the increase of the other probabilities associated with the processes involved in producing energy for the system:

```
System.runFeederAgent
Feeder.rxSignalUpTransducer
Feeder.rxSignal
Feeder.energyDistribution
```

Second, we should also note that, now that having an offspring comes at a high cost, the probabilities associated with reproducing are lower than 0.5 rather than higher (as noticed in the experiments presented in the previous section).

The zigzag shape of the population growth graph in Figure 30 is clearly due to the long period between re-seeding the habitat with food. The probability of running the `System.apoptosis` process (see Table A.1) may be used to calculate the average life span of an agent after the system stabilizes and this seems to be around 557 cycles. If instead of a period of 5000 cycles we use a period of 50 cycles we ensure that during an agent's life time the food in the environment is replenished more than once. Of course the 100 time reduction in the food replenishing period is complemented by a 100 time reduction in the energy contents of the food which will now be in the range [30, 40]. Simulations performed using these parameters have not resulted in stable systems for either 100 or 1000 initial agents. However if we double the energy contents of the food signals to a uniform random value in the range [60, 80] we do indeed obtain a stable system as shown in Figure 31 after 20000 cycles. Although the system has started with 1000 agents there was a moment during the simulation, as indicated by the arrow in Figure 31, when the population had dropped to a very low level.

Figure 31: The evolution of population size with 1000 initial agents and flooding every 50 cycles, energy contents [60, 80].

Out of 1000 agents there was one agent only that was able to survive and produce viable offsprings in the given habitat conditions. This situation is captured in Figure 32 and Figure 33 showing the distribution of agents after 500 and after 1000 cycles.

Figure 32: The distribution of agents after 500 cycles.

The arrow in Figure 32 points to the initial agent that gave birth to the only line of agents capable of surviving in this habitat. This agent is gone by cycle 1000 as shown in Figure 33.

Figure 33: The distribution of agents after 1000 cycles.

After 1000 cycles the population has dwindled down to about 118 agents however more than half of them belong to the robust lineage capable of surviving in this habitat with food replenishing every 50 cycles. The average alleles values at the end of simulation is given in Table A.2 in the Appendix.

When the food available in the environment is parceled in smaller amounts at smaller intervals, the agents should increase the frequency of scanning the environment for food signals and therefore run the `System.rxSignalUpTransducer` process with a higher frequency.

Also the energy obtained from the food signals should be distributed to the `System` and `Reproductive` agents faster because, if the energy reserves drop below 1 the processes involved with receiving the energy signals and replenishing the energy reserves will not be able to run and the agents while alive are not functional! These remarks are confirmed by an increase in the probability of running the processes `System.rxSignalUpTransducer` as well as `Feeder.txSignal` and `Receiver.rxSignal` (`System.rxSignal` probability was already quite high),

while the other Feeder processes involved in the energy extraction and distribution chain remained around 0.6.

5.7 Stochastic versus Non-stochastic Processes

In this section we use the model introduced in the previous section to study the effect of the stochastic component of the agents on the population's evolution. To do this we repeat the test but this time we initialize the probabilities of all processes to 1 except the apoptosis process which is initialized to 0.001 (the average probability reached at the end of the previous simulation). With these changes we started the simulation and very quickly, after 125 cycles almost the whole cellular space was filled with agents. As shown in Figure 34 most of the agents are first four generation agents (including the initial ones).

Figure 34: The distribution of non-stochastic agents after 125 cycles.

At that point it clearly looked like the non-stochastic agents were move active and they also produced a stable habitat. However, as time progressed after a while the population started to decrease and the simulation was stopped after 3000 cycles as shown in Figure 35.

Figure 35: The growth of a population of non-stochastic agents.

In fact an analysis showed that no more new agents have been created after cycle 233. To understand what went wrong with the population of non-stochastic agents we need to review the conditions under which an agent becomes sterile.

1. The Reproductive sub-agent is dead as a result of Reproductive.apoptosis process being executed. In real life this corresponds to a fatally diseased reproductive system. The organism as a whole continues to function but it cannot reproduce itself.
2. The Feeder sub-agent is dead and consequently the organism cannot process food and produce the energy required for the other processes to function, including the reproductive processes. In real life this situation corresponds to a fatally diseased digestive system.
3. The energy reserve of the reproductive sub-agent is so depleted that even the basic functions required to process the energy carrying signals cannot be performed. In real life this situation corresponds to a serious malfunction of the reproductive systems.
4. The energy reserve of the System sub-agent is depleted and the System agent cannot execute primary functions like: giving birth to a new agent (that it may carry in its signal space), receiving new energy signals from the feeder or "ingesting" new food

signals from the environment. This would correspond to other serious system malfunctions in real-life organisms.

To verify if the population of non-stochastic agents has been indeed afflicted by any of these diseases we have selected 5 different generations of agents as shown circled in Figure 36.

Figure 36: The distribution of non-stochastic (sterile) agents after 2000 cycles.

The "dissection" performed on these agents showed the following findings:

Red agent: the Reproductive sub-agent was found dead/null (case 1 above)

Blue agent: the System and Reproductive sub-agent energy reserves were < 1 (case 3 and 4)

Green agent: Feeder sub-agent is dead (case 2)

Black agent: the System and Reproductive sub-agent energy reserves were < 1 (case 3 and 4)

Turquoise agent: the System energy reserves were < 1 (case 4)

Luck had it that all four possible causes of sterility described above were found amongst the five agents analyzed. How did the habitat get into that situation? Why didn't we see this

happening with the stochastic agents? To answer these questions we have to analyze the special case when the agents filled up the whole habitable space as shown in Figure 34. When the whole habitable space is (almost) filled up with agents on the average there is just one food signal per agent per 50 cycles. This food signal contains an average of 70 energy units which get distributed (evenly) between the 3 sub-agents (System, Reproductive and Feeder) so each energy reserve gets replenished by about 23 energy units. However, each sub-agent has 5 processes that require 1 energy unit to execute and the probability for these processes to be executed (if there is energy) is 1 because we are analyzing the non-stochastic case. The result of this situation is that one or more sub-agents will run out of energy reserves and will become inoperable.

The estimated energy required by an agent can be calculated with the formula:

$$E(e) = \sum_{i \in I} p_i \text{ where } p_i \text{ is the probability that process } P_i \text{ to be executed and } I \text{ is the set of all the}$$

processes of a given sub-agent excepting the apoptosis process which does not require any energy to execute. Using the values in Table A.2, in Table 22 we present the average energy consumption of the sub-agents of a stochastic process compared to a non-stochastic process (X is the quantity required for producing a clone which is the same for both cases).

Table 22: A comparison of the energy requirements for stochastic and non-stochastic agents.

<i>Sub-agent</i>	<i>Stochastic</i>	<i>Non-stochastic</i>
System	2.36	6
Reproductive	1.99+X	5+X
Feeder	2.99	5

The unexpected answer to the previous questions is that what doomed the society of non-stochastic agents is their inefficiency! The stochastic agents do the same thing with less. They are more efficient and consequently survive better in difficult conditions.

Would the situation be different if the population would not get to a point where it totally fills in the cellular space? Wouldn't that be a more stable state? To answer this question we have performed a second experiment where the probability of running the apoptosis process had been set to 0.0084 (which was the starting value we used in many experiments with stochastic processes) rather than 0.001. The result of this experiment is shown in Figure 37.

Figure 37: The growth of a (initial) population of 1000 non-stochastic agents with apoptosis probability set at 0.0084.

This experiment shows clearly that while during the simulation the agents filled up to a maximum of 60% of the cellular space in the end the outcome was the same. The habitat was not sustainable and the whole population died after approximately 1000 cycles.

These two experiments showed the superiority of the stochastic model in producing a dynamic and viable habitat.

5.8 Mutation versus Random Alleles Setting

In this section we evaluate the contribution of the mutation operator versus randomly resetting the values of the alleles every time an offspring is generated. And again we start with the last habitat model presented in the "Autopoietic Habitats" section. Just like in the previous section we set all the probabilities of the apoptosis processes to a fixed value of 0.001. The values for all the other alleles are randomly generated every time rather than using the mutation operator

on the inherited set of alleles. The transformation mechanism was disabled in the first experiment and re-enabled in a second experiment. The results of the two experiments presented in Figure 38 and Figure 39 are almost identical showing that, unlike the case when mutation was used, the habitat is not sustainable and a slight initial increase in population is followed by a continuous decline in population and eventual extinction.

Figure 38: The growth of a (initial) population of 1000 stochastic agents with random offspring gene setting and transformation turned off.

After 3000 cycles with transformation turned off we found a population of 116 agents and 94 agents with transformation turned on. The transfer of genetic material did no make any significant difference.

Figure 39: Population growth with transformation turned off.

6 EMERGENCE IN STOCHASTIC HABITATS

"Creation is great, and cannot be understood"—Thomas Carlyle

In this chapter we discuss some very important characteristics of the Stochastic Habitats as emphasized through experiments performed using an enhanced SH model.

6.1 Enhanced Stochastic Agent Model

The experiments performed in the previous chapter have been performed using a simplified form of Stochastic Agent. In this section we introduce an enhanced model that will be used throughout the rest of this chapter.

The main phenotypic component of the Stochastic Agent is the group of sub-agent processes that run in parallel and their associated probability of running. A characteristic of the model used in the previous chapter is that most of the phenotypic process descriptors are just copies of identical descriptors making up the genotype (the chromosome). The only exception to this rule was the process `Feeder.energyDistributionGenesis()` which is a development process. A problem related to that approach is that the process parameters (see equation(4.3)) could not be changed, therefore depriving the model of an important mechanism of evolution. We share the opinion of other researchers that phenotypes of increasing complexity require indirect encodings and a strong artificial embryogenesis process (Stanley and Miikkulainen 2003). The enhanced SH model has a complete development phase and the associated chromosome is described in Table 23.

A second major difference between the two models is the way the offsprings are born. While in the model described in the previous chapter the offspring was created through cloning and it was immediately functional (after the release in the environment) the offsprings in the enhanced model have to go through a development process during which they can still receive energy signals from the parent through a special link to the parent which is severed

when the offspring is released in the environment. When the development phase is completed a signal is sent to the parent who then proceeds with expunging the offspring.

Table 23: The genotype of the enhanced stochastic agent.

Sub-agent	Process	Type	Parameters	
			Prob.	SS dist.
System	apoptosis	phenotype	1.0	
	createOrgans	stem	URnd(0,1) ¹	
	createFeeder	stem	URnd(0,1)	
	createWomb	stem	URnd(0,1)	
	createSignalingSpace	stem	URnd(0,1)	
	createTxRxProcesses	stem	URnd(0,1)	
	createIntakeFoodProcess	stem	URnd(0,1)	
	createTxRxUpTransducer- Processes	stem	URnd(0,1)	
	createTxRxUpTransducer- ToParent	stem	URnd(0,1)	
	runFeeder	phenotype	URnd(0,1)	
	runWomb	phenotype	URnd(0,1)	
	txSignal	phenotype	URnd(0,1)	
	rxSignal	phenotype	URnd(0,1)	
	txSignalUpTransducer	phenotype	URnd(0,1)	
	rxSignalUpTransducer	phenotype	URnd(0,1)	
	rxFoodFromEnvironment	phenotype	URnd(0,1)	
	txSignalToParent	phenotype	URnd(0,1)	
	rxSignalFromParent	phenotype	URnd(0,1)	
	createSystemDistributor	stem	URnd(0,1)	
	systemDistributor	phenotype	URnd(0,1)	
	wombEnergyDistribution- CreatorCreator	stem	URnd(0,1)	
	apoptosis	phenotype	1.0	

¹URnd(0,1) - Uniformly distributed random number in the interval (0,1)

Reproductive	createSignalingSpace	stem	URnd(0,1)	
	createTxRxProcesses	stem	URnd(0,1)	
	createTxRxUpTransducer- Processes	stem	URnd(0,1)	
	reproductiveDistributor	stem	URnd(0,1)	
	createProduceClone	stem	URnd(0,1)	
	produceClone	phenotype	URnd(0,1)	
	developOffspring	phenotype	URnd(0,1)	
	rxSignal	phenotype	URnd(0,1)	
	txSignal	phenotype	URnd(0,1)	
	rxEnergyUpTransducer	phenotype	URnd(0,1)	NRnd(.5,.1) ²
	rxSignalUpTransducer	phenotype	URnd(0,1)	
	txEnergyUpTransducer	phenotype	URnd(0,1)	
	txSignalUpTransducer	phenotype	URnd(0,1)	
	wombEnergyDistribution- Creator	stem	URnd(0,1)	
Feeder	apoptosis	phenotype	1.0	
	createSignalingSpace	stem	URnd(0,1)	
	createTxRxProcesses	stem	URnd(0,1)	
	createTxRxUpTransducer- Processes	stem	URnd(0,1)	
	createRxFood	stem	URnd(0,1)	
	createEnergyDistribution	stem	URnd(0,1)	
	feederDistributor	phenotype	URnd(0,1)	
	foodProcessing	phenotype	URnd(0,1)	NRnd(.5,.1)
	rxSignal	phenotype	URnd(0,1)	
	txSignal	phenotype	URnd(0,1)	
	rxFood	phenotype	URnd(0,1)	NRnd(.5,.1)
	rxSignalUpTransducer	phenotype	URnd(0,1)	
	txSignalUpTransducer	phenotype	URnd(0,1)	
	rxEnergyUpTransducer	phenotype	URnd(0,1)	
	txEnergyUpTransducer	phenotype	URnd(0,1)	NRnd(.5,.1)
	rxFoodUpTransducer	phenotype	URnd(0,1)	NRnd(.5,.1)

²NRnd(.5,.1) - normal distribution random number , mean=0.5, std.dev.=0.1

The advanced model uses three sub-agents just like the simpler model: System, Feeder and the Reproductive sub-agent but uses two types of processes as may be seen in Table 23. The processes labeled 'stem' are involved only during the agent development phase. Once they fulfill their purpose they stop running and are removed from the running list. The processes of type 'phenotype' may be modified by the 'stem' processes during development before being placed on the phenotype running list. Being part of the phenotype (as their name indicates) they run continuously (until the agent dies).

To understand how it works we have to discuss the various functional signaling chains and this task is helped by Figure 40 and the following description of the structure of the advance model.

Figure 40: Enhanced model of a stochastic agent.

As shown in Figure 40, a stochastic agent houses three major structures, access to an External Signaling Space (XSS), an Internal Signaling Space (ISS) and the Organs, all represented in Java as Hashtable objects. Both the XSS and the ISS can house multiple signaling space (SS) objects. This is another departure from the simpler model used in the previous chapter (as described in Figure 17). In the case of the System sub-agent, the external space is the ISS of the Reproductive sub-agent of the parent, during the offspring development, and the local habitat environment after its birth. A signaling space object within the XSS represents a 'window' or a 'contact point' within the external space. In biological terms a SS may simply correspond to a hole in the membrane of a cellular organism, for example, that may be used for signal exchanges between the cell and its environment.

The 'Organs' structure accommodates other sub-agents. The initial chromosome (see Table 23) has two genes; `System.createFeeder` and `System.createWomb` which when expressed will result in a Feeder sub-agent and a Reproductive sub-agent being added to the organs structure and a `System.runFeeder` and a `System.runWomb` process being added to the list of processes associated with the System phenotype. However, through mutations or crossover some agents may end up with multiple Reproductive or multiple Feeder sub-agents. It is clear that the architecture of the new model allows evolutionary changes that were not possible in the simple model used in the previous chapter.

Both the Reproductive and the Feeder sub-agent have a structure similar to the System sub-agent. They both have an XSS, an ISS and an Organs structure. The XSS of all sub-agents housed by the Organs are mapped onto the ISS of the System sub-agent as shown by the dotted lines in Figure 40. While the Organs structure of the Feeder sub-agent is empty (and consequently is omitted in Figure 40) the Organs structure of the Reproductive sub-agent is used to house the offsprings during their development stage and has been re-labeled 'Offspring pool' in Figure 40. The XSS of all the offsprings' System sub-agents are mapped onto the ISS of the Reproductive sub-agent during development and this offspring-parent link allows energy signals to flow from the parent to the offspring and feed its development. This relation is very important from an evolutionary point of view.

Through evolution the complexity of the artificial organism may increase and therefore the development phase may also increase in length. If the offspring is given only some fixed energy quantity, at some point during evolution this energy may not be enough to sustain the increased complexity and therefore hinder and eventually stop the evolution. By ensuring the offspring development phase is performed while the offspring is within and linked to the parent the new model fulfils an additional condition for unbounded evolution.

We can identify four signaling chains driving the functionality of the stochastic agent.

1. **The food chain:** "external food signal -> system internal signal space -> feeder internal signal space -> update of the acquired energy counter".
2. **The energy chain:** "convert the acquired energy to equal size energy signals -> feeder internal signal space <-> to system internal signaling space <-> to reproductive internal signaling space <-> to developing offsprings system internal signaling space". In addition to the main chain some local branches are used to take the energy signals from the internal signaling space and feed the correspondent sub-agent. The double arrow signal '<->' suggests the connection is bidirectional and the signals may travel back.
3. **The start reproduction chain:** "create offspring signal -> system internal signal space -> reproductive sub-agent internal signaling space -> add offspring to the pool of offsprings and give it a unique ID".
4. **The offspring birth chain:** "offspring finishes development and sends a signal to the parent including its ID -> parent reproductive sub-agent internal signal space -> offspring development completed signal is received by the reproductive sub-agent which removes the offspring from the offspring pool, includes it in a special signal of NONSELF type -> parent reproductive sub-agent internal signaling space -> parent system sub-agent internal signaling

space -> offspring released in one of the signaling spaces of the external signaling space of the parent, i.e. the environment".

A characteristic of all signaling chains is that signals travel from the ISS of a sub-agent to the ISS of another sub-agent. If an ISS contains a single signaling space only multiple chains may end up using it. For example food and energy signals may be mixed together if Feeder's ISS contains a signaling space only. Similarly, an even worse situation can occur in the System sub-agent if its ISS would contain a signaling space only because food, energy, create offspring and offspring carrying signals would be all sharing the same signaling space. Depending on how the signal handling functions are implemented, the mixture of signals may pose no problem or it may pose very serious difficulties for the agent's functionality. For example if we randomly select a signal for processing the probability of retrieving a specific kind of signal depends on the percentage of the types of signals in the mixture of signals sharing the signaling space. It is clear that a mechanism for creating additional signaling spaces tied to the processing of a certain type of signals could offer a greater potential to the evolution process.

A third difference between the enhanced model and the simpler model is given by the introduction of a second process descriptor parameter (see Table 23) that we call the 'signaling space distance'. The signaling space distance parameter has been created to facilitate the process of creating distinct signaling spaces and it will be described in more detail in the next section.

6.2 Signaling Space Differentiation Process

The 'signaling space distance' parameter is used in the development process of differentiating the signaling spaces used by different processes. The introduction of an additional signaling space is not a simple thing to do. Just simply creating an additional signaling space and adding it to the ISS of a sub-agent is of no effect if the processes that extract and process signals are not modified to use the newly introduced signaling space. Moreover, everything has to be done dynamically such that it may occur during the evolution process.

The conditions we have just outlined suggest that the identity of the signaling space must be supplied as a parameter to the method using it, on the one hand, and distinct methods must be used for processing different types of signals on the other hand. This last condition in particular is another departure from the simple model where all the signals that could occur in a signaling space were processed with the same method.

To understand how the new signaling spaces are created in the enhanced model we are going to take the food signaling chain as an example and show how the processes associated with it as well as the signaling spaces associated with these processes are created during the development phase.

The creation of a chain of processes that has two ends, like the food processing chain that starts in the environment and ends with the food processing method within the Feeder, has two obvious solutions and we have decided upon a solution that starts with a development process in the Feeder and goes all the way up.

When during the development phase, the `Feeder.createRxFood()` stem process is executed it will result in the following actions:

1. The chromosome is scanned and all the genes for `Feeder.rxFood()` phenotypic processes are collected in one list (although Table 23 shows only one gene of each kind through mutations we may actually have multiple genes of the same kind). To the signaling space distance (let's name this variable `ssd`) we apply the 'ceiling' mathematical function to create an integer and use this integer as a key to retrieve a signaling space from the ISS Hashtable. If there is no such signaling space (the object retrieved is `null`) we create one and place it back in the ISS using the key derived from `ssd`. In mathematical terms this can be expressed as $key = \lceil ssd \rceil$. The `Feeder.rxFood()` has two parameters: the probability of running (which is left untouched) and the key of the signaling space that must be used by the `rxFood()` process to retrieve the food signals from. This second parameter is replaced with

the key that was generated as shown above. Also, the key is added to a list of keys to be used at step 2. All `rxFood()` processes are added to the Feeder running list.

2. The chromosome is scanned and all the genes for `Feeder.rxFoodUpTransducer()` phenotypic processes are collected. The task of this process is to move food signals from XSS (which is mapped onto the ISS of the System sub-agent) into an ISS signaling space and they have three parameters: the probability of running, the key of the ISS SS and the key of the XSS SS. The key for the ISS SS is randomly selected from the list of keys prepared at step 1 (!) thus linking an `rxFoodUpTransducer()` process and an `rxFood()` process into a food processing chain. The key for the XSS SS is generated from the `ssd` of the gene currently processed as shown at step 1 and an appropriate SS is created with the generated key if it doesn't already exist. After the `rxFoodUpTransducer()` arguments are generated the processes are added to the Feeder running list and the keys are added to another list of keys identifying SS in the System ISS to be used at step 3.
3. All the genes for the `System.createFoodIntakeProcess()` stem processes are expressed. These processes have two arguments: a probability of running (that will be left unchanged) and a list of keys which is set for all of the processes to the list created at step 2. All modified processes are added to the running list of the System sub-agent.
4. `Feeder.createRxFood()` stem process is completed and may be removed from the Feeder running list.
5. When a `System.createFoodIntakeProcess()` is executed (remember that it was placed on the System running list at step 3) it will collect from the chromosome all the genes for the `System.rxFoodFromEnvironment()` phenotypic processes and modify their parameters as follows. The probability of running is left intact, the key of the ISS SS is randomly selected from the list of keys supplied as parameter at step 3 and an XSS SS key is generated from the gene's `ssd`. An SS with the generated key

will be created in XSS if it doesn't already exist. All `rxFoodFromEnvironment()` processes are added to System's running list and `createFoodIntakeProcess()` stem processes may be completed and removed from the System running list.

Because the *ssd* variables are initialized using a normal distribution random variable with mean 0.5 and standard deviation 0.1 they will all convert to *key* = 1 and all the ISS structures for all the sub-agents will contain a single SS only. However, during evolution, the *ssd* variables may be mutated (just like the probabilities of running) using the formula: $ssd+ = gauss(0,1) * ssd * 0.1$. When, through mutations, the value of an *ssd* variable becomes greater than 1, a *key* = 2 will be generated during development and a part of the food chain will start to be diverted through a different signaling space.

In conclusion, it is important to notice that two aspects are instrumental in the process of signaling space (or path) differentiation: the signaling space distance evolvable variables and the development process.

6.3 Apoptosis and Life Expectation

The fourth major difference between the model introduced in this chapter and the simpler one used in the previous chapter relates to how the apoptosis function is implemented. As may be seen in Table 23, there is an apoptosis process for each sub-agent. In the simpler model the apoptosis process was implemented as a process running with a fixed very small but constant probability. The advanced model that we described in this chapter uses an apoptosis implementation that is much closer to the theoretical model introduced in chapter 4. Therefore, the implementation of the apoptosis process collapses the three sub-components described in the theoretical model (the inhibitor, the accelerator and the terminator) in one process with equivalent functionality as shown by the pseudo code in Table 24. The equivalent of the inhibitor process is given by the event that the sub-agent energy reserve contains enough energy to cover the 'quanta' apoptosis toll (*quanta* = 1 for all experiments except those in the last section) in which case the apoptosis probability increases by a 1.01 factor. In case that such an energy is not available the apoptosis probability will be increased by a 1.1

factor. If a randomly generated number is smaller than the probability of apoptosis the terminator is applied and the sub-agent is killed.

Table 24 : Enhanced model apoptosis process pseudo code.

```

if (uniform-random-number < probability-of-apoptosis)
    kill sub-agent
else if (sub-agent energy reserve > quanta) {
    subtract quanta from sub-agent energy reserve
    probability-of-apoptosis *= 1.01
}
else
    probability-of-apoptosis *= 1.1

```

We compared the more sophisticated apoptosis method with the simpler one in a series of experiments. All the experiments have been performed using the enhanced model except the apoptosis method which was set to the simpler apoptosis method with a constant probability of running (and therefore dying) of 0.001 for some experiments and to the more sophisticated method with a starting probability of apoptosis set to 1.0E-7 for others.

Figure 41: Life expectancy using the simple apoptosis model.

In Figure 41 we show the result of the first experiment performed using the simpler apoptosis method with a fixed (0.001) apoptosis probability and a starting population of 5000 agents.

In Figure 42 we show the result of the second experiment, the population decline, using the advanced apoptosis model and a starting population of 5000 agents.

Figure 42: Life expectancy using an advanced apoptosis model.

A first comparison of the two curves shows that while for about 300 cycles the population of agents using the advanced apoptosis method had barely declined the population of agents using the simpler method has already lost almost 1500 agents. On the other hand while by cycle 1500 all the agents using the advance apoptosis method are gone, over 1000 agents using the simpler method were still alive. In fact some agents using the simpler apoptosis method were still alive after 8000 cycles (although some of their organs may have failed!).

How do these models compare with life expectancy curves in real life populations?

To answer this question we have resorted to the Gompertz law that is used in actuarial science for specifying a simplified mortality law (Kenney & Keeping 1962). Using $s(x)$ as the probability that a newborn will achieve age x , the Gompertz law is given by the equation:

$$s(x) = \exp[-m(c^x - 1)] \quad (6.1)$$

for $c > 1, x \geq 0$ (Gompertz 1832). Using the values $c=1.01$, $m=0.000007$ the scaled (by 5000) Gompertz curve is depicted in Figure 43.

Figure 43: Life expectancy according to the Gompertz law.

While the curve in Figure 43 does not resemble the curve in Figure 41, it is in fact similar with the part of the curve in Figure 42 that starts around cycle 500 until the end. A significant population drop after 300 cycles causes the difference that we see in Figure 42. With the exception of the apoptosis process probability of running which was set to 1 and not modified (not to be confused with the probability of apoptosis, i.e. the probability of dying, which was set to $1.0E-7$ and modified as explained in the pseudo-code in Table 24) all the other processes had their probability of running randomly initialized.. Therefore, the drop in population relates to the fact that because of the random initialization certain agents were not viable. If an agent happened to be non-functional, its probability of apoptosis would increase at a higher rate (1.1

instead of 1.01) than a functional process! While this fact does not affect the agents using the simpler apoptosis method it does affect the agents using the advanced apoptosis method. On the other hand, the Gompertz law describes the mortality law in a population of **functional** organisms.

To correct this discrepancy we used the method described in the previous chapter of averaging the alleles of the agents surviving more than 500 cycles and ran a simulation where **all** the agents have been initialized using these averages (we do not give these averages because we do not plan to make any comments regarding individual alleles values).

In Figure 44 we show the mortality curve in a population of 5000 agents initialized as specified superimposed with the Gompertz law.

Figure 44: Life expectancy of agents using advanced apoptosis versus the Gompertz law

This time the two curves are almost identical. The small drop of about 250 agents after 300 cycles is explained by the fact that the average genotype of functional agents does not necessarily produce a functional genotype in all habitat conditions (like crowding).

The result that the mortality rate in a population of stochastic agents follows the Gompertz law, a law applied to living organisms is remarkable.

If the mortality curve of a population of agents using the new apoptosis mechanism is so close to a Gompertz curve why not simply use the Gompertz law as given by equation 5.1? There are two reasons for using our method rather than using the Gompertz equation. First, our method is more efficient from a computational point of view. Second, and more importantly, our method is tightly coupled with how efficient an organism is while Gompertz's law, just like the simple apoptosis method, is an abstract law that does not mirror the efficiency of an organism in its life expectancy. This second difference is fundamental.

The theoretical life expectancy of an agent using the simple apoptosis method is $1 / p_{at}$ and for $p_{at} = 0.001$ the life expectancy is 1000 cycles. The life expectancy of the agents using the

new apoptosis method can be calculated using the formula $\frac{\sum_t k(t) * t}{\sum_t k(t)}$ where $k(t)$ represents

the number of agents that died at cycle t . If we apply this formula to the data collected in the previous three experiments we obtain the results shown in Table 25.

Table 25: The life expectancy of the agents in the three experiments performed

Population	Life expectancy
Simple apoptosis method	1012
Advanced apoptosis, random initialization	527
Advanced apoptosis, functional agents	1081

While it is nice that one apoptosis method closely follows a well known theoretical law, what impact does it have on the population's chances to survive? To answer this question we revisited the first two experiments after enabling the reproduction process.

As it is shown in Figure 45, the population of agents using a simple apoptosis method, although benefiting from a longer life expectancy is almost extinct after 6000 cycles.

Figure 45: Population of agents with simple apoptosis after 6000 cycles.

On the other hand the population of agents using the advanced apoptosis method after 6000 cycles (see Figure 46) has succeeded in creating two growing clusters of viable agents.

Figure 46: Population of agents with advanced apoptosis after 6000 cycles.

The result of these last two experiments shows clearly that the advanced apoptosis mechanism helps evolution in two ways. First and foremost it does not indiscriminately kill agents as the simpler method does. There is a strong link between agent fitness and life expectancy when the advanced apoptosis method is used. Second, the agents that are not fit are killed quicker and their resources returned to the environment for the other agents to use.

6.4 Evolution and Adaptation in Stochastic Habitats

In this section we plan to show the results of our study of the stochastic agents' capacity to adapt and evolve to changing environmental conditions.

The experiments have been performed using the following setup. The initial population of 1000 agents was randomly initialized with exception of the apoptosis process which has been set to run with probability 1 while the apoptosis probability (i.e. the probability of running the 'terminator' process in the theoretical model) was set as before to 1.0E-7.

In the first group of experiments we have used only mutation. The various mutation operators are described in Table 26. A parent's mutated chromosome is placed in the offspring's System "empty shell", so to say, and during the development phase the phenotype is created. So the mutated chromosome has a direct impact on the development of the offspring as well as its functionality.

Table 26: Mutation operators used in the enhanced model

Mutator	Applied to	Operation
Gene mutator	Probability of running	$p = p + \text{Gauss}(0, 0.1) * p$
	Signaling space distance	$ssd = ssd + \text{Gauss}(0, 0.1) * ssd$
Chromosome mutator	Chromosome	Duplicate a gene
		Remove a gene

The gene mutator is applied with probability 1.0E-5 to the probability of running section of the genes while ssd mutator and the chromosome mutator are applied with probability 1.0E-3. The reason for the higher ssd and chromosome mutation is that we wanted to speed up the mutations directly impacting an agent's development and phenotypic functionality.

Each cell of the 2D 100x100 grid is initialized with a food signal with a nutrient value between 1000 and 1200 energy units which is also the value used for initializing the agent's total energy reserves. When an offspring is created the energy required for initializing the offspring energy reserves must be covered by the reproductive sub-agent. If enough energy to cover these requirements is not available the reproduction cannot take place. The food is replenished periodically by planting new food signals in those cells that do not have one. Please note that if an agent dies it will also leave an extra signal food in the cell it occupied while alive. This is the only scenario under which certain cells may contain 2 food signals. The first experiment uses a food replenishing schedule as described in Table 27.

Table 27: Food replenishing schedule for the first experiment

Time	0	2000	3500	5000	10000	20000
Replenish Period	10	50	100	200	400	STOP

Figure 47: Agent distribution after 2000 cycles and color coding scheme.

In Figure 47 we show the distribution of agents after 2000 cycles. With numbers we designate the areas of interest showing small clusters of agents with structural mutations.

The color coding for 4th (and subsequent) generation agents is also shown in Figure 47. Using Repast capabilities and specially developed code we probed the agents for functional mutations they had undergone during evolution and summarized the results in Table 28. The column labeled 'Function' shows if the respective gene fulfils a positive role (+), a negative role (-) or has no role (0). A negative role would be if the gene translates into a function that consumes energy but it doesn't have any contribution to agent's functionality. It may be seen that genes with both negative and neutral functions have been left in the chromosome on purpose to verify how the evolution will deal with them. The letter S or P indicates whether the gene will express into a stem or phenotypic process. A number in the 'Zone' columns indicates the chromosome has so many genes of that kind as the number indicates. A '-' sign indicates the gene is missing and a 'd' indicates a mutation suffered by gene's ssd. A clear cell indicates that the respective gene has suffered no mutations.

Table 28: Experiment 1: summary of agent mutations after 20000 cycles.

No	Function	Gene	Zone					
			1	2	3	4	5	6
0	P+	S.apoptosis						
1	S+	S.createOrgans						
2	P+	S.growFeeder						
3	P+	S.growWomb						
4	P+	S.growSignalingSpace						
5	S+	S.createTxRxProcesses						
6	S+	S.createIntakeFoodProcess						
7	S+	S.createTxRxUpTransducerProcesses						
8	S+	S.createTxRxUpTransducerToParent						
9	P+	S.runFeeder						
10	P+	S.runWomb		2				
11	P+	S.txSignal						
12	P+	S.rxSignal						
13	P+	S.txSignalUpTransducer						

14	P-	S.rxSignalUpTransducer		
15	P+	S.rxFoodFromEnvironment		
16	P+	S.txSignalToParent		
17	P+	S.rxSignalFromParent		
18	S+	S.createSystemDistributor		2
19	P+	S.systemDistributor		
20	S+	S.wombEnergyDistribCreatorCreator		
21	P+	R.apoptosis		
22	P+	R.growSignalingSpace		
23	S+	R.createTxRxProcesses		
24	S+	R.createTxRxUpTransducerProcesses		
25	P+	R.reproductiveDistributor		
26	S+	R.createProduceClone		
27	P+	R.produceClone		2
28	P+	R.developOffspring		
29	P0	R.txSignal	2	2
30	P+	R.rxSignal		
31	P+	R.txEnergyUpTransducer		
32	P+	R.txSignalUpTransducer		2
33	P+	R.rxSignalUpTransducer		
34	P+	R.rxEnergyUpTransducer		
35	S+	R.wombEnergyDistributionCreator		
36	P+	F.apoptosis		
37	P+	F.growSignalingSpace		
38	S-	F.createTxRxProcesses		
39	S+	F.createRxFood		
40	S+	F.createTxRxUpTransducerProcesses		
41	S+	F.createEnergyDistribution		
42	P+	F.feederDistributor	d	d
43	P+	F.foodProcessing		
44	P0	F.txSignal		
45	P0	F.rxSignal		
46	P+	F.rxFood		
47	P+	F.txSignalUpTransducer		
48	P+	F.rxEnergyUpTransducer		
49	P+	F.txEnergyUpTransducer	d	
50	P+	F.txSignalUpTransducer		
51	P-	F.rxFoodUpTransducer		

Among the mutations summarized in Table 28 of more interest are: 1. In Zone 1 agents suffered a differentiation of the signaling space used by the energy path, at the System level

and consequently a separation of the energy signals from the rest of the signals received by the System sub-agent. 2. Zone 2 agents suffered a doubling of the gene `S.runWomb()` that translates in the Reproductive sub-agent being run twice every cycle. 3. Zone 5 where a doubling of the `S.systemDistributor()` gene resulted in the System agent absorbing more energy signals (comparable to a doubling of the blood vessels capacity in biological terms).

Figure 48 shows the distribution of agents after 3500 cycles (using the schedule in Table 27).

Figure 48: Experiment 1: agent distribution after 3500 cycles

The summary of the mutations with functional implications is given in Table A.3 in the Appendix. It is interesting to notice from the beginning that all agents in Zone 2 from Figure 47 are gone. Running the Reproductive sub-agent twice has become very expensive when the food replenishing period has dropped from 10 to 50 cycles and the agents paid the price.

A second interesting observation is that two different clusters of agents (Zone 1 and 8) were now missing gene 48, a phenotypic process bringing energy signaling signals back to Feeder (it has a positive function). When `F.txEnergyUpTransducer()` process is executed a certain percentage of the energy signals from the Feeder SS attached to is transported to a System SS. The `F.rxEnergyUpTransducer()` transports a certain percentage of those signals back to the Feeder. The agents in two different clusters have suffered a mutation that removes the feedback link from the System to the Feeder therefore guaranteeing a better supply of energy to the System but more important to the Reproductive sub-agent and the offsprings which are further down in the energy chain. Clusters 2 and 3 have experienced mutations on gene 49's `ssd` leading to a differentiation of the System SS used by the energy chain.

Figure 49 shows the distribution of agents after 5000 cycles and the summary of the functional mutations are given in Table A.4.

Figure 49: Experiment 1: agent distribution after 5000 cycles

Between cycle 3500 and cycle 5000 the experiment has used a food replenishing period of 100 cycles (10 times slower than at the beginning). Clusters 1 and 3 shown in Figure 28 have continued to grow all the way to cycle 5000. Notable mutations are shown by clusters 4 and 5 (Figure 30) where agents experienced a doubling of gene 18 and 35 respectively. These two genes are linked to two stem processes. A doubling of such genes translates in two copies of the respective processes being run and consequently a higher probability that the phenotypic characteristics triggered by the respective genes would be created faster. In a nutshell these kinds of mutations simply speed up the development process.

Figure 50 shows the distribution of agents after 10000 cycles and the summary of the functional mutations are given in Table A.5.

Figure 50: Experiment 1: agent distribution after 10000 cycles.

An analysis of the mutations summarized in Table A.5 shows a remarkable thing.

Except cluster 8, the agents in all the other clusters are missing gene 31. When expressed, gene 31 leads to the formation of a phenotypic process, `R.txEnergyUpTransducer()`, carrying energy signals from the Reproductive sub-agent back to the System sub-agent. The absence of this process simply blocks the energy signals from going back to the System, once they have entered the Reproductive sub-agent. This mutation ensures a better energy supply to the Reproductive sub-agent and offsprings under increasingly difficult feeding conditions.

The agents in cluster 8 found a different solution to increasingly difficult environmental conditions. The doubling of the genes 1, 20 and 26, all three linked to stem processes simply ensures a much faster development process and therefore the production of viable offspring more efficiently. The 4th mutation in the chromosome of the agents in cluster 8 is a doubling of gene 32 linked to the phenotypic process `R.txSignalUpTransducer()`. A higher probability for this process to run translates in signals from the Reproductive sub-agent being transferred up to the System faster. Such a signal is 'Reproductive sub-agent development is completed', a signal also linked to a faster development process phase. Another type of signal from the Reproductive sub-agent to the System is the signal carrying the developed offsprings ready to be released in the environment. Therefore the doubled gene also leads to a faster release of those offspring that finished the development phase.

Figure 51 shows the distribution of agents at the end of simulation (after 20000 cycles) and the summary of the functional mutations are given in Table A.6.

The absence of gene 31 that we've talked about before continues to be prevalent in the population (11 clusters out of 12 probed showing this mutation). Other interesting mutations are related to saving energy by eliminating genes linked to phenotypic processes with negative roles. This is the case of gene 14 missing in agents in clusters 1, 2 and 4. Also gene 38 is missing from agents in clusters 4 and 8. Gene 38 is linked to stem processes and because it is missing two phenotypic processes are not created. However these are the processes expressed by genes 44 and 45 and as shown in all the summary tables these processes have neutral roles and therefore they may be eliminated with no repercussions.

Figure 51: Experiment 1. Agent distribution after 20000 cycles

Figure 52 showing the population growth during the experiment 1 indicates population declines linked to food replenishing slowdowns after 3500, 5000 and 10000 cycles followed by recoveries, as a consequence of the adaptation to new environmental conditions.

Figure 52: Population growth during experiment 1.

How much did the structural mutation operators contribute to the agents' capability to adapt to environmental changes? The second experiment addresses this question by using exactly the same food replenishing schedule used in the first experiment but disabling the chromosome mutator and the gene's ssd mutator (to prevent signaling path differentiation). The population growth curve is presented in Figure 53.

Figure 53: Population growth during experiment 2

By the end of the simulation the population, although not extinct, had dwindled down to 80 agents. This experiment comes to show that the mutation operators structurally impacting agent's phenotype are essential in adaptation.

A third experiment tried to further differentiate between the contributions made by the chromosome mutation operator and the gene's ssd mutation operator, both capable of inducing structural changes. In this experiment the gene's ssd mutation operator has been disabled while the other mutation operators were enabled.

The population growth curve for the third experiment is presented in Figure 54. This experiment shows that in spite of a severe decline in population after the food replenishing period dropped to 400 cycles, the population has eventually stabilized.

Figure 54: Population growth during experiment 3

The fourth experiment, just like the third one, tries to differentiate between the contribution made by the chromosome mutation operator and the gene's ssd mutation operator. In this experiment the chromosome mutation operator has been disabled while both gene mutation operators have been enabled. The population growth curve for the fourth experiment is presented in Figure 55.

Figure 55: Population growth during experiment 4

The outcome of the experiment 4 was similar to that of experiment 2, although this time at the end of the simulation the number of agents was a little higher, 135. It shows that the signaling space differentiator (ssd part of the gene) did not make enough of a contribution to allow the population to survive under the most stringent environmental change.

In the fifth experiment we disabled the operator performing mutations on the probability of running part of the gene but enabled the gene's ssd and the chromosome mutator. The population growth curve for the fifth experiment is presented in Figure 37.

Figure 56: Population growth during experiment 5

Figure 56 looks almost identical to Figure 52. This experiment shows very clearly that the two mutation operators that could affect the phenotypic structure are much more powerful than the mutator operating on the processes' probability of running and are responsible for agent's capability to adapt to environmental changes.

A second interesting aspect is that at the end of the fifth simulation almost all the agents had a square shape (see Figure 57) indicating that they had suffered signaling path differentiation mutations. The summary of the structural mutations given in Table A.7 shows that all 10 clusters probed had differentiated the energy signaling path at the System sub-agent level.

A third interesting observation is that, unlike experiment 1, there were no missing genes. The agents' successful adaptations had been achieved only through gene doubling (there is also 1 triple) and energy signal path differentiation at the System level.

Figure 57: Agent distribution and clusters probed at the end of fifth experiment

In the next two experiments we study the importance of gradual environmental changes as opposed to abrupt changes. Except for the schedule the other conditions are identical to the ones used in experiment 1 (therefore all mutation operators are enabled).

In the sixth experiment we set the food replenishing period to 400 cycles from the beginning. Not unexpectedly the population completely died after about 3700 cycles.

In the seventh and last experiment the schedule was modified as follows. In the first 3500 cycles we used an abundant feeding schedule with food being replenished every 10 cycles. Starting at cycle 3500 until the end we have abruptly switched to the very scarce feeding

schedule with the replenishing period set to 400 cycles. The population growth curve for the fourth experiment is presented in Figure 58.

Figure 58: Population growth during experiment 7

The outcome of this experiment was somehow surprising. At the end of the experiment the population dropped to 13 agents. Correlated with the first experiment, this last experiment shows the importance of gradual, as opposed to abrupt, environmental changes. When the schedule was changed at cycle 3500 the population was about 4500 agents, higher than the number of agents in experiment 1 after 10000 cycles when the replenishing period was also changed to 400 cycles. Although starting with a larger population the last experiment had a totally different outcome than experiment 1, underscoring the crucial importance of the rate of environmental changes relative to the population's rate of adaptation.

We conclude by summarizing the findings of the evolutionary experiments discussed in this section.

- The stochastic agents are very adaptable to various environmental changes

- Amongst the operators used, the mutation operators that impact agent's phenotypic structure (the chromosome mutator and the gene ssd mutator) have the highest impact on the agent's ability to adapt.
- There is a clear correlation between the agent's ability to adapt and the rate of environmental change. Experiment 1 showed that a period of about 8000 cycles of gradual changes was needed to adapt to the harshest environment conditions. Considering the average life time of functional agents to be around 1000 cycles, we may say that it takes about 8 generations of agents going through gradual changes to adapt to the most difficult environmental conditions.
- Abrupt environmental changes seem to lead to population extinction.
- Through adaptation and natural selection different "creatures" have emerged from the original ones all providing distinct and innovating solutions to the same problem.

These results indicate that our model may be successfully extended and used as a basis for simulating natural systems.

6.5 The Effect of Genetic Material Exchange

In our studies in this chapter we have used only mutation operators. However, it is well known that the crossover operator also helps the process of finding and incorporating good "building blocks" into the genotype. As we have shown in the previous chapter, the same goal of transferring genetic material between individuals may be achieved using transformation, a mechanism employed in prokaryotes. In the experiments performed with the simple model and discussed in the previous chapter the contribution made by transformation to agent's ability to adapt was not significant. So we asked ourselves the question whether it would make a difference in the framework of the experiments performed in this chapter with the advanced agent model.

To answer this question we have made a little change to the mechanism for transferring genetic material from an agent that has died to an agent that is still alive. If an agent scheduled to die does still contain a functional Reproductive sub-agent, we create a signal agent that carries a randomly selected fragment of the dying agent's chromosome. Unlike the chromosome fragment carrying signals created in the previous chapter that were static (left behind in the cell where agent's death occurred) and immortal, the signals that we create in this enhanced model are dynamic and they are mortal, in one word, they are Signal agents (or messenger-agents in the terminology used in Chapter 4).

Using as basis the apoptosis mechanism described at the beginning of this chapter we set the apoptosis probability of the created signal agents equal to the apoptosis probability of the Reproductive sub-agent from which we have retrieved the chromosome fragment. If the agent that died was an old agent its Reproductive sub-agent, even if still alive, most likely is also old and it will transfer its "age" to the Signal agent that we create. Because the Signal agent does not have a mechanism for absorbing energy from the environment its apoptosis probability will always increase by a factor of 1.1 (instead of 1.01 that corresponds to a slower rate). This mechanism ensures the chromosome fragments carrying signals have a finite life and will be eliminated from the environment, usually in a few hundred cycles. A second aspect of these signals is that they are moving around. Every cycle, the signals may move from their previous location to any other location in a Moore neighborhood. These two changes are fundamental.

Using the same initial population and the same setup as in the previous section we performed another experiment with all genetic operators as well as the transformation mechanism turned on. The population growth curve for this experiment is shown in Figure 59 and the agent distribution is shown in Figure 60. The population growth during the first 12000 cycles is almost identical to that shown in Figure 54 (experiment 3) in the previous section but the curve shows lower peaks than the curve shown in Figure 52 for experiment 1. We should recall that the experiment performed in this section mirrors experiment 1 performed in the previous section with the exception that now transformation is turned on. So it seems that in the first half of the experiment transformation did not have a positive effect which is not

necessarily surprising considering that the genetic material incorporated in live agents belonged to agents that have died and one would assume that inability to adapt could have caused an earlier death in some agents.

Figure 59: Population growth with all mutation operators and transformation turned on.

The real surprise came in the second part of the experiment, during very harsh environmental conditions when the growth experienced after the 12000 cycles beats all previous results.

The agent distribution shown in Figure 60 is very interesting. While in the lower half of the grid we see agents and patterns very similar to those seen in Figure 57 (for example), in the upper part of the grid (and wrapping around at the very bottom left of the figure) we see a special kind of agent density that we have never seen before. The probing of these agents (see summary of mutations in Table A.8 in the Appendix) did reveal some quite unexpected structural mutations. The agents in clusters 1, 2 and 9 show a very high number of gene 5 (S.createTxRxProcesses) and 48 (F.rxEnergyUpTransducer). Gene 5 corresponds to an stem process involved in the creation of two processes amongst which only one is important. A high number of this gene will simply translate in those processes being created faster. Also unremarkable is the high number of gene 48. In fact a probing of that gene shows that was carrying a very low probability of running. Multiple versions of gene 48 simply raised the

probability of the `F.rxEnergyUpTransducer` to run. On the other hand the agents in the lower half of the figure were characterized by gene 31 missing and the energy signaling path differentiation characteristics to the adaptation discussed in the previous section.

Figure 60: Agent distribution and clusters probed

In fact the adaptation of the agents in clusters 1, 2 and 9 was so good they were able to reproduce even towards the end of the 400 cycle food replenishing period. So what may have possibly caused this unusual ability to adapt?

The answer to this question is the genetic material exchange made possible by transformation. Transformation not only allows for one genetic material exchange to occur. In fact the mechanism allows an exchange of genetic material to occur **repeatedly**. In other words, one live agent may process multiple chromosome fragments carrying signals in its life time, leading to significant genetic changes. A consequence of this is the emergence of a special kind of agents that rely on well tuned combinations of probabilities of running allowing them to survive and even prosper under the most severe conditions. It is also clear that structural mutations have also helped the process.

To prove this point we have performed a second experiment mirroring the second experiment performed in the previous section. We kept intact the food replenishing schedule and disabled the chromosome mutator and the gene's ssd mutator (to prevent signaling path differentiation) but kept enabled the probability of running mutator. The population growth curve for the second genetic material exchange experiment is shown in Figure 61.

Figure 61: Population growth with one gene mutator and transformation turned on.

When we compare the curve shown in Figure 61 with Figure 53 it becomes clear that transformation has helped the population survive the most severe feeding schedule. In fact the growth shown just towards the end of the experiment continues, and after 40000 cycles the population reached about 2000 agents. In experiment 2 described in the previous section the population was almost extinct after 20000 cycles.

The two experiments described in this section show unequivocally that genetic material exchange improves an organism's ability to adapt in stochastic habitats. The mechanism that we described for transferring genetic material between agents is very powerful when realistically implemented. As we have mentioned this mechanism is actually used in nature by prokaryotic organisms.

6.6 A Repair Mechanism in Stochastic Agents

At the beginning of this chapter we showed how stem processes are involved in creating phenotypic processes during development. We also mentioned that these processes, once they executed (completed), were removed from the running list of the phenotype.

A very interesting, simple and natural extension of the stochastic agent model may be obtained by simply **not removing** the stem processes after being executed once, leaving open the possibility for them to be executed again. If the stem processes are designed such that they kick in if and only if the processes they are supposed to create are missing, what we automatically get (for free) is not only a process that can be used during development, but also a process that can be used during the lifetime of the agent to repair other damaged processes. The natural emergence of this **repair mechanism** comes to show once more the power of the artificial embryogenesis process implemented in our SH model.

However, in the model described and used in the previous sections we didn't need a repair mechanism because apoptosis had been implemented only at sub-agent level. To test the effect of the repair mechanism we implemented the apoptosis mechanism at the process (both stem and phenotypic) level and changed the energy tax associated with it from 1 unit to 0.1 units applicable every cycle at running time. This may now be interpreted as a metabolic penalty paid at each process level. When the metabolic tax cannot be paid the penalty is a faster increase rate in the apoptosis probability of that process.

When apoptosis kicks in the process is simply considered completed and subsequently is removed from the running list. If that process is a phenotypic process, the stem process in charge of its creation, if still functional, will detect its absence and recreate it by completing the repair mechanism. We would like to point out that certain stem processes may also be the result of a development process in which case the repair mechanism would work pretty much the same way. While the recreation of certain processes may be relatively inexpensive from an energetic point the repair of a sub-agent (a Feeder or a Reproductive sub-agent) comes at a high price that must be paid by the System.

To study the impact of the repair mechanism we ran two experiments, one with the repair mechanism turned off and another one with the repair mechanism turned on, and built the life expectancy curves for two populations of 5000 agents. The two curves together with the Gompertz theoretical curve for $m=0.00002$ and $c=1.0095$ are shown in Figure 62.

Figure 62: Life expectancy curves: with repair mechanism turned off, turned on, and the theoretical Gompertz law curve.

This time before doing the experiments we have performed a pre-run, with reproduction enabled, during which we have selected a healthy agent (based on its number of offsprings and the crowdedness factor of its habitat area). The two experiments that we have performed have used the genetic makeup of this individual rather than the average of a number of healthy individuals as we have previously done. As a consequence of this approach the population drop previously experienced has been eliminated.

The repair mechanism made possible by an extension of the artificial embryogenesis process of the enhanced stochastic agent's model proved to be very effective in extending the lifetime of a good number of individuals while the maximum age range remained the same. The

closeness between the life expectancy curve when the repair mechanism was turned on and the Gompertz theoretical law is a remarkable result.

7 DATA CLASSIFICATION IN COMPUTATIONAL HABITATS

"And still be doing, never done"—Samuel Butler

The data classification algorithm that we present in this chapter is the first algorithm developed using the Computational Habitat model. The approach behind this algorithm has been worked out between (Tulai & Oppacher 2004b) and (Tulai & Oppacher 2004c). As a consequence of that, this algorithm that we are going to call **CH**-based data Classifier (CHC) shares many characteristics with MSUV, the algorithm presented in (Tulai & Oppacher 2004b) and with MSWV, the algorithm presented in (Tulai & Oppacher 2004c). In the following sections we briefly review CHC's characteristics and how they differ from MSUV and MSWV.

7.1.1 Representation and Species Definition

CHC uses the same representation and species definition as MSUV and MSWV. The computational component of CHC agents are also decision trees with internal nodes carrying Boolean valued attributes and leaf nodes carrying one of two possible class labels. The lack of a good genotypic representation for decision trees has resulted in a departure from the **CH** model. Therefore, the decision tree representation gives the genotype and the phenotype of the computational component just as in MSUV and MSWV. However, this is an issue we plan to review in our future work.

When it comes to the size of the i -th species population, CHC uses a fixed population size (all species have the same size), similar to MSUV rather than the automatic species dependent population size calculation (see equation (3.5)) used by MSWV. Consequently, when comparative results will be presented later on we shall give the population size used by CHC and MSUV.

CHC uses the same mutation operators as MSUV and MSWV.

7.1.2 Rule Discovery and Automatic Speciation

To understand how the rule discovery and automatic speciation mechanism works we will use Figure 63 which shows a theoretical distribution of agents in the rule fitness landscape.

Figure 63: A theoretical distribution of agents in the rule fitness landscape.

In Figure 63, the different size hills represent various decision rules. The smaller hills correspond to **weak** decision rules while the taller hills correspond to **strong** decision rules, as discussed in section 3.2.2. The white dots represent the agents which, of course, have different fitness. Because the decision trees are randomly initialized the distribution of agents in the fitness landscape is random. The agents with the lowest fitness are shown on the floor of the fitness landscape. These agents are not committed to any decision rule yet. Other agents are shown placed on different hills at various heights. Ideally we would like to make sure that all the decision rules are **completely** discovered.

For an agent to completely discover a decision rule is equivalent to climbing all the way to the peak of the hill. Two major issues come to mind by analyzing Figure 63.

- How to make sure that all (or most) of the decision rules are discovered
- How to make sure that they are discovered as fast as possible

MSUV (Tulai & Oppacher 2004b) has answered the first issue by placing each individual in a separate deme from the beginning and allowing the selection operator to function only between the parent and its child and therefore we call it **restricted selection**. If instead of restricted selection we place all the agents in a single population and allow them to compete freely, one can immediately see from Figure 63 that, if at some point during the evolution most of the agents will be close to the peak of hills A and B, the decision rule represented by hill C may be abandoned. Even if certain agents would have reached the peak of hill C (such that the rule is completely discovered) they may still have a lower fitness than agents involved in discovering rule A. MSUV uses only restricted selection to answer the first major issue but did not address the second major issue.

To understand the second issue, on the hill labeled B we have shown two pairs of agents linked by arrows. The start of an arrow marks a parent agent and the tip of the arrow marks an offspring. If each agent is placed in a separate deme, as argued above, the offspring in the pair labeled 1 will replace its parent while the offspring in the pair labeled 2, having a lower fitness than its parent, will be eliminated. However, it is clear that preserving both the parent and the offspring of the second pair and dumping both agents of pair 1 would accelerate the process of climbing the fitness hill labeled B. From this point of view it is clear that it would be better to have all the agents **climbing the same hill** placed in the same deme and to let them compete freely with each other, accelerating the rule finding process.

CHC and MSWV use restricted selection at the start of the algorithm but also have added automatic speciation to detect and cluster (in the same deme) agents that climb the same hill in

order to speed up the rule discovery process, therefore addressing the second major issue. How does the automatic speciation work?

The Computational Habitat model offered a different environment facilitating a new way of thinking. All the agents of a species are placed equidistantly in the 2D cellular space. The training instances are treated as food signals. Each agent has access to all training instances available at one point in time (the next section will deal with the issue of instances in detail). If an agent correctly identifies a training instance it may consume the energy it contains. Afterwards each agent sends a summary of the food items that it considers edible (which corresponds to the training instances, whether positive or negative that it correctly identified) together with its own (x, y) coordinates to the other agents in a Moore neighborhood of a radius proportional to its own energy reserve which is proportional to the number of edible food signals (i.e. to the number of training instances correctly identified). This mechanism is very similar to the mechanism used in (Tulai & Oppacher 2004a) for selecting mating partners. Each agent processes the received messages and tries to find another agent that has a menu of edible items that includes its own and if it finds one, takes a step towards it. If the other agent is already near it (in a Moore neighborhood of radius 1) the current agent simply moves into the second agent's cell. The selection operator is restricted to the occupants of a deme.

This automatic speciation mechanism (the agents sharing a deme are treated as the same subspecies and are allowed to freely compete, and, in future extensions to the algorithm, mate) has been discovered in the CHC algorithm and become an integral part of MSWV system.

This natural mechanism inspired by the Computational Habitat model has the advantage that speciation takes place continuously and un-programmed when it takes place within the model. Unfortunately MSWV could not use a continuous speciation and had to resort to a programmed speciation, for two reasons:

- When performed within the structure of MSWV the speciation is more intensive computationally

- Agents with a very low fitness (the uncommitted agents distributed on the landscape floor in Figure 63) tend to have a very small number of items on their "edible" list and get **instantly** gobbled up by other agents when speciation is performed within MSWV. CHC doesn't have this problem because agents move by 1 step towards other agents rather than merging instantly as in MSWV.

The automatic speciation that CHC has introduced helped addressing the second major issue and it has significantly improved MSWV as shown in (Tulai & Oppacher 2004d).

One issue that future research in Computational habitats has to address is the "motivation" of agents for developing the mechanism that allows the automatic speciation. It is fair to say that in CHC, although inspired by the CH model, the mechanism did not emerge but was rather designed into the computational sub-agent.

7.1.3 Training Data

One significant difference between CHC and the two algorithms that it bridged, MSUV and MSWV, is linked to the training data. While both MSUV and MSWV use all the data for training, CHC divides the training data into chunks of roughly equal size and trains for a small fixed number of generations on each chunk before moving to the next one. CHC has generally used a "chunking" factor such that the number of instances in each chunk was 50 to 100 (see Table 29).

To make a parallel with nature, the agents in CHC stay for a certain period of time (cycles or number of generations) on a certain field providing a certain list of foods before moving on to another field, and so on, coming back full circle to the first field. The reason for this approach is that we have been searching for a local algorithm that could be implemented within the *CH* model and this approach seemed more natural.

7.1.4 Rule Confidence and Rule Weight

The rule confidence mechanism has been introduced in MSUV and it is shared by all 3 algorithms. On the other hand, the rule weight mechanism has been introduced only in

MSWV to allow the system to deal with unknown feature values. Therefore CHC (just like MSUV) does not have the rule weight mechanism that MSWV has and cannot handle datasets with unknown values. The comparison between the three algorithms will have to use datasets without unknown values.

7.1.5 Fitness Calculation

The equation for individual fitness calculation, as described in section 3.3.2.5 contains the rule weight (or path weight) pW . However, if the classification is performed on datasets without unknown values, the rule (path) weight will always be 1. Therefore, on datasets without unknown values, all three algorithms will use the same fitness calculation method.

7.1.6 Reward Mechanism Changes

CHC has introduced some changes to the reward mechanism, changes inspired by nature and by the *CH* model. Because certain training instances are harder to classify than others (usually those instances that are close to the plane separating different classes) we have introduced a mechanism meant to reward agents a little more for correctly identifying training instances that are harder to identify. Instead of keeping the basic reward for recognizing one instance fixed (equal to 1) we are lowering the reward by a very small factor after an agent has recognized that particular instance. This way, it is clear that instances that are easier to recognize will offer a smaller and smaller reward from agent to agent. This mechanism seems to favor the agents that get trained first but this should be of no concern because the populations are always reshuffled before each training step.

For the experiments presented in section 7.1.9 we have used a reward adjustment factor of 0.02 and because the number of training instances in one chunk was kept around 100, the reward could reach a minimum value of around 0.8 (instead of 1). This mechanism may parallel a natural situation where a source of food is shared by multiple organisms with a presumable decrease in the nutrient value of the source from the first organism using it to the last one.

7.1.7 Training Time

MSUV uses a fixed training time, usually 100 generations. CHC has experimented with a species dependent training time mechanism but it has used the training accuracy as a stopping criterion which is a less sensitive parameter (because only the fittest individual of a deme votes) than the population average fitness that MSWV has finally used. Nevertheless, it was CHC that has first experimented with this very important issue that eventually helped MSWV reduce the number of parameters uses.

However in the experiments discussed later CHC will use a fixed training time, just like MSUV

7.1.8 Testing

In all three algorithms testing is done based on a voting system which theoretically relies on equations (3.8) and (3.9) for predicting the class of a given test sample. However, because MSUV lacks the automatic speciation mechanism and each species has the same number of individuals the factor $M / \text{card}(M_i)$ is not necessary and therefore is missing while CHC does use it. As previously mentioned, for datasets without unknowns, $pW = 1$, so the rule weight factor is not used by either MSUV or CHC.

7.1.9 Experimental Results

In Table 29 we summarize the comparison between the three algorithms on a group of 5 datasets.

Table 29: Accuracy and dataset evaluations (in thousands) for MSUV, CHC and MSWV

ID	MSUV				CHC				MSWV			
	PS	TT	Acc.	NE	PS	TT	K	R	Acc.	NE	Acc.	NE
brw	100	100	97.20°	20	100	100	9	10	94.69	2.2	96.91°	78.54
bpa	200	100	65.30	40	100	280	9	20	66.12	3.1	67.39°	112
cmc	200	500	53.60°	300	100	260	13	20	49.33	6	52.99°	234.34
gls	100	400	67.67°	240	100	420	9	30	64.02	28	72.01°	69.62
pmi	100	100	75.30	20	100	100	10	10	75.87	2	74.61°	177.14

The mnemonics used for various columns heading have the following meaning:

- ID – the dataset (for full name and properties see Table 9 on page 64).
- PS – population size (number of agents in each species)
- TT – total training time (in generations)
- Acc – testing accuracy (given as % of total number of instances)
- NE – total number of dataset evaluations (in thousands) calculated as follows (nC is the number of classes in each dataset taken from Table 9).
 - For MSUV: $PS * nC * TT$
 - For CHC: $PS * nC * TT / K$
 - For MSUV: $\sum_{i=1, nC} PS_i * TT_i$
- K – number of chunks the dataset is divided into (valid just for CHC)
- R – number of generations the fractional dataset instances are evaluated before we Rotate to another dataset chunk (valid just for CHC)

The CHC results presented in Table 29 are not great. Just in one case, the Pima dataset, CHC was able to show better results than the other two algorithms. In addition to this, although what we show in Table 29 are the average results of 10 10-fold cross validation runs, for the CHC we have generally picked the best from a pool of runs with various K and TT parameters.

However, what makes CHC still interesting and worth further research is the small number of dataset evaluations which translates in a shorter running time. In all cases the number of

dataset evaluations for the CHC algorithm is smaller than for the other two algorithms, and in many cases it is orders of magnitude smaller. Because Evolutionary Computation methods are generally slow, EC algorithms that are faster are worth investigating.

As discussed in Chapter 2, many previous Artificial Life systems have tried to use their models for information processing or for optimization (problem solving) and for most of them the success has been very limited (for example the Avida model or the Pushpop system) or of a theoretical nature (SR loops or P-systems). We consider that the first application of the Computational Habitat model to a complex task like data classification is a successful attempt to make an Alife system do something useful not only because of the reasonable quality of the results (especially the short execution time) but especially for the insights that we gained doing it.

8 LONG-TERM GOALS AND CONCLUSIONS

"I never think of the future--it comes soon enough"-- Albert Einstein

Research is a never ending process, so the last chapter comes to summarize the accomplishments of this thesis following an outline of the short and long term extensions to the lines of research contained in this thesis.

8.1 Extensions to MSWV System

MSWV is a new algorithm compared to other very well known classification algorithms, like C4.5, PART, NB, etc. While the results presented in section 3.3.3 clearly prove that, with the exception of SMO, MSWV outperforms all other classification algorithms, we firmly believe that it may be further improved. In this section we present two lines of research.

8.1.1 Data Driven Initialization

In section 7.1.2 we used Figure 63 to illustrate a theoretical distribution of agents in the rule fitness landscape and help our explanation of how the idea of automatic speciation "emerged" in the context of *CH*. In the same section (as well as section 3.3.2.1) we mentioned that the decision trees are randomly initialized and therefore the agents are randomly distributed over the rule fitness landscape.

If the result of the classification process depends on the initial "coverage" of the decision rules with agents, it is clear that a random initialization does not guarantee an optimal coverage for a given number of agents. Moreover, a data driven initialization, seems, at least intuitively, to provide a better initialization method.

During initialization three things are randomly generated;

- The topology of the decision trees

- The feature number associated with each internal node
- The attribute function associated with each node

It is this last step that in the current version of MSWV is randomly performed that we believe may benefit from a data driven initialization. The decision trees would be randomly generated and to each node we would randomly associate a feature number (as the current algorithm does), but let the attribute function un-initialized. This partial random initialization would be followed by a data driven initialization step. Training instances and decision trees would be randomly selected from the available pool. If the selected decision tree can correctly classify the instance (that would happen if enough attribute functions have been initialized in the process) we consider this training instance covered and select another one. If it is incorrectly classified we select another decision tree. If it cannot be classified because not enough attribute functions have been initialized, we proceed with initializing the attribute function by constructing attributes based on the given instance. And so on.

It is conceivable that the first two steps of the initialization may also be performed with non-random method. For example, we may perhaps use some feature selection methods (Roubos et. al. 2003, Jain & Zongker 1997, Pudil et. al. 1994) to select the feature used (step 2) and also influence the topology of the decision trees (step 1).

8.1.2 Crossover Operator

In (Tulai & Oppacher 2004a) we proposed and used a crossover operator specifically designed for decision trees characterized by the restriction we impose on the internal nodes to be associated with distinct features (see section 3.3.2.1). To overcome this restriction the crossover operator has to ensure the two branches selected for crossover use the same subset of distinct features although they may be in a different order.

The crossover operator used, although efficiently implemented, is too restrictive as only branches using the same set of features may be swapped.

Using the same idea for subtree ID as in (Tulai & Oppacher 2004a), we may select branches for crossover operation such that the branch that is to be added to the tree should have no nodes in common with the rest of the tree (after the removal of the branch selected for crossover). If $ID(b)$ is the ID number of the subtree headed by the node b , the eligibility for applying crossover to two branches $b1$ and $b2$ from two separate trees with the root node of the first tree denoted by $r1$ may be simply checked using the relation: $(ID(r1) \otimes ID(b1)) \wedge ID(b2) == 0$ where ' $\otimes$ ' and ' $\wedge$ ' are the 'xor' and the 'and' bitwise operators.

The crossover operator may be applied after the automatic speciation mechanism kicks in to those demes containing more than one agent.

Because crossover is a very disruptive genetic operator a side effect of applying the crossover operator may be a total change in the topology and rule "coverage" of the offsprings which may force a continuous evaluation of the relationship between the agents within a given deme. This will negatively affect the algorithm's efficiency.

8.2 Future Research in *CH*

It is our firm belief that the Computational Habitats model has the potential to become a new domain of research and as such we cannot fully predict its future evolution, as one could not have possibly predicted how the research in Genetic Algorithms would evolve, forty years ago. So we will limit ourselves to simply expand the themes that we have developed in this Thesis and in the Thesis Proposal.

8.2.1 Simplified Model of Nature

The results of our research described and discussed in Chapter 5 and Chapter 6 have already given a strong indication of the significant potential carried by *SH* as a simplified model of nature.

However complex the *SH* agents may become through evolution, it is quite clear that the unbounded evolutionary capability (Bedau et. al. 1998) cannot be achieved without making the

first part of equation (4.3) a subject of evolution just as we have done with the second and third part. To put it into words, we must be able to evolve methods P_i and their encoded genotype representations $\mathbf{1}P_i$. In addition to supporting evolutionary changes the phenotypic processes P_i must also be dynamically modifiable as a result of internal and external interactions. Only then SH models will become perfect models of nature and have an unbounded evolutionary capability. Such an unbounded evolutionary capability would also include, for example, the capability to discover new genetic operators.

One immediate implication of this requirement is that in such advanced models agents will have to be interpreted rather than compiled and executed. Inevitably, such simulations will be extremely slow.

It is also conceivable that at that point the research in CH will benefit from the progress achieved in Evolutionary and Genetic Programming.

Just as in section 4.3 ("Formal Representation of Computational Habitats") we talked about using C, Lisp, Linda, NN etc. for representing the computational sub-agent in problem solving studies we also can talk about using any of these languages (and others) for complete agent representation in studies of emergence in SH as models of nature. And again a good choice may be offered by NN as they may be naturally interpreted, encoded and grown (Gruau & Whitley 1993, Gruau 1994, Köhn 1996, Stanley & Miikkulainen 2002).

8.2.2 Artificial Embryogenesis in Stochastic Agents

In their review of prior work in artificial embryogenesis Stanley & Miikkulainen mention two lines of research: the *grammatical approach* of Lindenmayer (Lindenmayer 1968) and the *cell chemistry approach* inspired by the work of Turing (Turing 1952). While it appears that the kind of AE employed by the stochastic agents described in Chapter 6 is closer to the cell chemistry approach (an approach that tries to mimic how physical structures emerge in biology) we would like to study the AE process of stochastic agents in the context of the taxonomy for AE defined in (Stanley & Miikkulainen 2003).

Therefore a special place in our future work is assigned to continuing and expanding the artificial embryogenesis model of stochastic agents.

8.2.3 *CH* as Environments for Problem Solving

Although in Chapter 7 we have shown how approaching data classifications within the *CH* framework has helped us in developing new and original concepts that later helped MSWV to become a world-class classifier, the *C* in *CH* has barely been touched.

In our future work we will continue to develop the CHC algorithm (for example coming up with a good genotype encoding and a proper AE process) and compare its performance to MSWV's performance. Continuing the lines of research outlined in (Tulai 2003) we plan to also study other classes of problems like bit-string and symbolic problems and expand the real-numbers class of problems (to which CHC belongs) to Multi-objective optimization.

Problems that are difficult to solve within the context of EC are of special interest to study using the *CH* model. The issue of "what makes a problem difficult for EC" has been addressed by the research community and over the years a list of such problems has been compiled and analyzed and difficulties in the search process identified and correlated with a list of characteristics (Vassilev et al. 2000, Goldberg et al. 1992, Naudts & Kallel 1998, Jones & Forrest 1995). The list of such characteristics would include: epistasis, fitness distance correlation, deception, scaling, noise, information content, and information stability and more recently symmetry (van Hoyweghen et al. 2002).

A long term objective is the study of learning in Computational Habitats.

8.2.4 Theoretical Development of the *CH* Model

In this thesis we have expanded the theoretical foundation of the *CH* model that was started in the thesis proposal. However, the theoretical framework must be further developed to allow us to make predictions about the kind of applications that could be efficiently approached within the *CH* model. We should also be able to describe what the limitations of

the model are, what problems cannot be solved or cannot be efficiently solved using the *CH* model.

A second fundamental question we want to answer is whether Computational Habitats can be tailored to certain classes of problems. If this question is answered positively we can talk about significant design time savings in applying already developed environments to certain classes of problems. Also if the answer is yes, we would like to define what the characteristics of such environments may be. Reusability, after all, is a very important practical problem.

8.3 Conclusions

Thus we conclude our journey from Genetic Algorithms to the concept of Computational Habitats.

Along the way we have made some forays into the study of coevolutionary relationships, speciation and parameter free evolutionary algorithms.

The nature of a relationship (be it competitive or cooperative) plays an important role in determining various components of the evolutionary model (like problem decomposition, credit assignment, etc.). We enhanced the classical cooperative coevolution model by replacing the new species introduction trial-and-error phase with a more scientific and robust method. In the new algorithm, called **Competitive-Cooperative Coevolution Genetic Algorithm** (CCC-GA), the newly introduced species are the winners of the competitive phase between sub-populations evolved with GAs using different strategy parameters. We showed the proposed method to be robust. The CCC-GA algorithm was used to find the best cascade neural networks evolved solution (to the two-spiral problem) reported in the literature.

The second contribution of this thesis is the **Multiple Species Weighted Voting** (MSWV) system. MSWV is a complex, genetics-based data classification algorithm that uses static and dynamic speciation and a sophisticated voting system that makes classification a collective task. Using thirty-six real-world learning tasks we show that MSWV significantly outperforms (over 97% confidence) a number of well known classification algorithms (NB, IB1, C4.5 and PART)

and is outperformed by SMO, an SVM based algorithm, with 55% confidence. When the comparison is done on a subset of 17 datasets containing at least one continuous numerical feature MSWV significantly outperforms all the classification algorithms it was compared to. MSWV is between 3 and 4 orders of magnitude slower than the other algorithms when the comparison is done on datasets with less than 1000 instances but it scales as $O(N)$ while SMO scales as $O(N^2)$ (see the discussion on speed and scalability in section 3.3.5). MSWV performs well on noisy datasets, can handle records with unknown values and does not require post pruning. Based on the experimental results we believe that at this moment MSWV is the best GBML system reported in the EC literature.

The third contribution of this thesis is our work towards **parameter-free Evolutionary Algorithms**. In the case of CCC-GA, neither the mutation nor the crossover probabilities are pre-determined (through prior experiments) but are rather selected (from a finite set of possibilities) as a result of the competitive-cooperative coevolution phase of the algorithm. MSWV may be seen as a parameter free GBML system. Although the algorithm has few parameters that may be modified all the results reported were obtained without modifying any of these parameters. We **generalized CCC-GA** by encoding the strategy parameters as chromosomes in every genome in the population, and having them evolve together with the application chromosomes. We showed that the generalized CCC-GA is robust and performs well on a test bed of seven popular continuous functions, and converges to the global optimum on the problems studied. The generalized CCC-GA performs equally well to the simple CCC-GA on the two-spiral problem.

The fourth contribution of this thesis is to introduce the concept of **Stochastic Habitat** and its extension, the **Computational Habitat** and place them on a theoretical foundation. In Chapter 2 we argued that deterministic Cellular Automata and Non-Uniform Cellular Automata are special cases of Computational Habitats. As generalizations of NUCAs and CAs, Computational Habitats on 1 or 2-dimensional cellular landscapes can also support universal computation.

The fifth contribution of this thesis is given by **the first practical implementations of a Stochastic Habitat** as well as the first results obtained within the framework of Computational Habitat. In this thesis we show that life expectancy of populations of stochastic agents closely follows the Gompertz theoretical curve for human populations. That, combined with the strong adaptive power showed by the stochastic agents, suggests that SHs could successfully be used as simplified models of nature.

I feel a journey is just as important as the goal waiting at the end of it, and I personally enjoyed every bit of my journey to the final goal, this PhD thesis.

Ottawa, April 11, 2004

BIBLIOGRAPHY

- Adami, C. (1995): Learning and complexity in genetic auto-adaptive systems, *Physica D*, 80, 154-170.
- Adami, C. (1998): *Introduction to Artificial Life*, Springer-Verlag New York.
- Adami, C., Brown, C.T., (1994): Evolutionary learning in the 2D artificial life system "avida". In R. Brooks and P. Maes, (Eds.), *Artificial Life IV* (377-381).
- Adamidis, P., Petridis, V., (1996): Co-operating populations with different evolution behaviors. In: *Proceedings of the IEEE International Conference on Evolutionary Computation*, Piscataway, NJ, USA , IEEE Press, 188-191
- Andre, D., Bennett, F.H., Koza, J.R., (1996): Evolution of intricate long-distance communication signals in cellular automata using Genetic Programming. In *Proceedings of the 5th Artificial Life Conference*.
- Aha, D., Kibler, D., Albert, M., (1991): Instance-based learning algorithms. *Machine Learning* vol. 6, 37-66.
- Back, T., (1991): Self-Adaptation in Genetic Algorithms. *Proceedings of the 1st European Conference on Artificial Life*, MIT Press 263-271.
- Banâtre, J.-P., Le Métayer, D., (1996): Gamma and the chemical reaction model: ten years after. In: *Coordination programming: mechanisms, models and semantics*, World Scientific Publishing, IC Press.
- Banzhaf, W., Dittrich, P., Rauhe H., (1996): *Emergent computations by catalytic reactions*, *Nanotechnology* 7, 307 – 314.
- Barendregt, H.P., (1992): Lambda Calculi with Types. *Handbook of logic in computer science*. Oxford U. Press. chapter 13, 117-309.
- Barendregt, H.P., Barendsen, E., (1991): *Introduction to lambda calculus*. Technical report, Department of C.Sc., Catholic U. of Nijmegen, The Netherlands.
- Bedau, M.A., Snyder, E., Packard, N.H., (1998): A classification of long-term evolutionary dynamics. In C. Adami, R. Belew, H. Kitano, and C. Taylor (Eds.), *Artificial Life VI* (Cambridge: MIT Press), 228-237

- Bernadó-Mansilla, E., Garrell-Guiu, J.M., (2003): Accuracy-based learning classifier systems: models, analysis and applications to classification tasks. *Evolutionary Computation* 11(3) 209-238.
- Berry, G., Boudol, G., (1992): The Chemical Abstract Machine. *Theoretical Computer Science*, vol. 96, 217-248.
- Byl, J., (1989): Self-Reproduction in Small Cellular Automata. *Physica D*, 34, 295-299.
- Cain, M.L., Damman, H., Lue, R.A., Yoon, C.K., (2000): *Discover Biology*. Sinauer Associates, Sunderland MA, USA.
- Calude, C.S., Paun, G., (2000): Computing With Cells and Atoms in a Nutshell. *Complexity*, Vol. 6(1) 38-48.
- Carriero, N., Gelernter, D., (1989): Linda in Context. *Communication of the ACM*. Vol. 32(4).
- Carvalho, D.R., Freitas, A.A., (2002): A genetic algorithm with sequential niching for discovering small-disjunct rules. *Proceedings of Genetic and Evolutionary Computation Conference*. Morgan Kauffman publishers, San Francisco California.
- Cattal, R., Oppacher, F., Deugo, D., (2001): Supervised and unsupervised data mining with an evolutionary algorithm. *Proceedings of the Conference on Evolutionary Computation*, Volume: 2 (767 -774).
- Cech, T.R., (2000): The ribosome is a ribozyme. *Science*, 289:878-9.
- Channon, A.D., Damper, R.I., (1998): Evolving novel behaviors via natural selection. *Proceedings of Artificial Life VI*.
- Channon, A.D., Damper, R.I., (2000): Towards the evolutionary emergence of increasingly complex advantageous behaviours. *International Journal of Systems Science*.
- Chou, H.H., Reggia, J.A., (1997): Emergence of self-replicating structures in a cellular automata space. *Physica D*, 110, 252-272.
- Chou, H.H., Reggia, J.A., (1998): Problem solving during artificial selection of self-replicating loops. *Physica D*, 115, 293-312

- Cohen, P.R., (1991): A Survey of the Eighth National Conference on Artificial Intelligence: Pulling Together or Pulling Apart? *AI Magazine*, 11(4) 16-41. Technical Report 91-68, Dept. of Comp.Science, University of Massachusetts
- Cristianini, N., Shawe-Taylor, J., (2000): *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, Cambridge, U.K.
- Dittrich, P., Ziegler, J., Banzhaf, W., (2001): Artificial Chemistries - a review. *Artificial Life*, 7(3):225-275.
- Dorigo, M., Di Caro, G., (1999): The Ant Colony Optimization metaheuristic. *New ideas in optimization*, McGraw Hill, London UK, 11-32.
- Dorigo, M., Stützle, T., (2001): The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. *Handbook of metaheuristics*.
- Fahlman, S.E., Labiere, C., (1990a): The cascade correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania
- Fahlman, S.E., Labiere, C., (1990b): The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, CA. 524-532.
- Ferber, J., (1999): *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. Harlow: Addison Wesley Longman.
- Fontana, W., Buss, L.W., (1994): The arrival of the fittest: toward a theory of biological organization. *Bulletin of Mathematical Biology*, 56, 1-64.
- Fontana, W., (1997, 1991 reprint): Functional Self-Organization in Complex Systems. Reprint in: *Pattern Formation in the Physical and Biological Sciences*, Addison-Wesley, Reading, MA, 43-63.
- Fontana, W. (1992): Algorithmic chemistry. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pp 159-210, Redwood City, CA. Addison-Wesley.
- Forrest, S., Jones, T., (1994): Modeling complex adaptive systems with Echo. In R. J. Stonier and X.H. Yu, editors, *Complex Systems: Mechanism of Adaptation*. IOS Press.

- Fournet, C., Gonthier, G., (1996): The reflexive chemical abstract machine and the join-calculus. *23-rd ACM Symposium on Principles of Programming Languages*.
- Fournet, C., Gonthier, G., Levy, J.J., Maranget, L., Remy, D., (1996): A Calculus of Mobile Agents. In: *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR)*, Springer LNCS 1119.
- Frank, E., Witten, I.H., (1998): Generating accurate rule sets without global optimization. In: Shavlik, J., editor, *Machine Learning proceedings of the 15th international conference*. Morgan Kaufmann 144-151.
- Friedman, J.H., (1996): *Another approach to polychotomous classification*. Technical report, Stanford department of Statistics [www-stat.stanford.edu/reports/friedman/poly.ps.Z].
- de Garis, H., (1990): Genetic Programming: Building Artificial Nervous Systems with Genetically Programmed Neural Network Modules. *Proceedings of the 7th International Conference on Machine Learning*.
- Gelernter, D., (1985): Generative communications in Linda, *ACM Transactions on Programming Languages and Systems*, 7(1) 80-112.
- Goldberg, D.E., (1989): *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, MA.
- Goldberg, D.E., Horn, J., Deb, K., (1992): *What Makes a Problem Hard a Classifier System?* IlliGAL Report No. 92007.
- Gompertz, B. (1832): On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. *Phil. Trans. Roy. Soc. London* 123, 513-585.
- Graham, L., Opacher, F., (2003): Trying to evolve sorting networks in echo. *Proceedings of the Congress on Evolutionary Computation*.
- Gruau, F., Whitley, D., (1993): *The cellular development of neural networks: The interaction of learning and evolution*. Technical Report 93-04. Ecole Normale Supérieure de Lyon. France.
- Gruau, F., (1994): *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD Thesis. Ecole Normale Supérieure de Lyon. France.

- Hallgrímsson, B., Willmore, K., Hall, B., (2003). Canalization, developmental stability, and morphological integration in primate limbs. Submitted to: *Yearbook of Physical Anthropology*.
- Hansen, L.K., Pedersen, M.W., (1994): Controlled growth of cascade correlation nets. In: *Proceedings of the International Conference on Artificial Neural Networks*, volume 1, Sorrento, Italy. 797-800.
- Harik, G.R., Lobo, F.G., (1999): A parameter-less genetic algorithm. In Banzhaf, W. et al. (Eds.), *Proceedings GECCO-99*: 258-267.
- Harvey, I., Husbands, P., Cliff, D., (1994): Seeing the Light: Artificial Evolution; Real Vision. *From Animals to Animats 3: Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior*, 392-401.
- Hausser, D., (1988): Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177-221.
- Holland, J.H., (1962): Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3), 297-314.
- Holland, J.H., (1986): Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In: Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Vol. 2). Morgan Kaufmann Publishers, Los Altos, CA.
- Holland, J., (1992): *Adaptation in Natural and Artificial Systems*, University of Michigan Press. 2nd edition.
- van Hoyweghen, C., Naudts, B., Goldberg, D.E., (2002): Spin-Flip Symmetry and Synchronization. *Evolutionary Computation*, 10(4), 317-344.
- Hsu, W. H., Gustafson, S. M., (2002): Genetic Programming and Multi-Agent Layered Learning by Reinforcements, *Proceedings of GECCO*, 764-771.
- Huberman, B.A., Hogg, T., (1992): The Emergence of Computational Ecologies. Santa Fe Institute *Lectures in Complex Systems*.
- Hutton, T. (2002): Evolvable self-replicating molecules in an artificial chemistry. *Artificial Life*, 8(4), 341-356.

- Ibanez, J., Anabitarte, D., Azpeitia, I., Barrera O., Barrutieta, A., Blanco, H., Echarte, F., (1995): Self-inspection based reproduction in cellular automata. In: *Advances in Artificial Life* F. Moran et al. (Eds.), LNAI-929, Springer-Verlag 564-576.
- Jain, A., Zongker, D., (1997): Feature Selection: Evaluation, Application, and Small Sample Performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(2): pp.55.
- John, G., Langley, P., (1995): Estimating Continuous Distributions in Bayesian Classifiers. In: P. Besnard and S. Hanks (Eds.) *11-th Annual Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Mateo.
- Jones, T. Forrest, S., (1995): Fitness distance correlation as a measure of problem difficulty using genetic algorithms. *Proceedings of the 6th International Conference on GA*.
- Kenney, J. F., Keeping, E. S., (1962): *Mathematics of Statistics*, Part 1, 3rd ed. Princeton, NJ: Van Nostrand.
- Kibler, D., Langley, P., (1988): Machine Learning as an experimental science. In: Sleeman, D., editor, *Proceedings of the Third European Working Session on Learning*, pp. 81-92.
- Köhn, P., (1996): Genetic encoding strategies for neural networks. IPMU, Granada, Spain.
- Laing, R., (1977): Automaton models of reproduction by self-inspection. *Journal of Theoretical Biology*, 66, 437-456.
- Langton, C.G., (1984): Self-reproduction in cellular automata, *Physica D*, 10, 135-144.
- Langton, C.G., (1986): Studying artificial life with cellular automata, *Physica D*, 22, 120-149.
- Langley, P. (2000): Crafting Papers on Machine Learning. In *Proceedings of the 17th International Conference on Machine Learning*. Stanford, CA: Morgan Kaufmann, 1207-1211.
- Lindenmayer, A., (1968): mathematical models for cellular interaction in development: Parts I and II. *Journal of Theoretical Biology*, 18, 280-315.

- Lobo, F.G., Goldberg, D.E., (2001): *The Parameter-less Genetic Algorithm in Practice*. IlliGAL Report No 2001022.
- Luksha, P.O., (2002): Formal Definition of Self-reproductive Systems, *Artificial Life VIII*, Standish, Abbass, Bedau (Eds.) MIT Press. 414-417.
- Luo, Y., Davis, D. N., Liu, K., (2001): Information and Knowledge Exchange in a Multi-Agent System for Stock Trading. *IEEE Communication Society and the International Engineering Consortium*, pp. 47-55.
- Marchand, M., Shawe-Taylor, J., (2003): The set covering machine. In: *The Journal of Machine Learning Research*, 3(4-5), 723-746. MIT Press, Cambridge, MA. USA.
- Maynard Smith, J., (1970): Natural selection and the concept of protein space. *Nature*, London 225, 563-564.
- Maynard Smith, J. & Szathmáry, E. (1995): *The Major Transitions in Evolution*. Freeman & Co., Oxford.
- Meleau, N., Dorigo, M., (2002): Ant colony optimization and stochastic gradient descent. *Artificial Life*, 8(2) 103-121.
- Milner, R., Parrow, J., Walker, D., (1992): A calculus of mobile processes, parts I and II. *Journal of Information and Computation*, 100(1):1-40, pp. 41-77.
- Milner, R., (1990): Functions as Processes. In: *Automata, Language and Programming*, 17th Int. Coll., vol 443 of LNCS, Springer Verlag, pp. 167-180.
- Mitchell, M., (2001): *An Introduction to Genetic Algorithms*, 7-th printing. MIT.
- Mitchell, M., Crutchfield, J.P., Hraber, P.T., (1994): Evolving cellular automata to perform computations: mechanisms and impediments. *Physica D*, 75, pp. 361-391.
- Morita, K., and Imai, K., (1996): Self-reproduction in a reversible cellular space. *Theoretical Computer Science*, 168, pp. 337-366.
- Morita, K., Imai, K., (1997): A simple self-reproducing cellular automaton with shape-encoding mechanism, *Artificial Life V*, MIT Press, 489-496.

- Mühlenbein, H., Paaß, G., (1996): From recombination of genes to the estimation of distributions I. Binary parameters. In *LNCS 1411*, pp. 188-197.
- Mühlenbein, H., Schlierkamp-Voosen, D. (1993): Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, 1(1). pp. 25-49.
- Mühlenbein, H., Schomisch, M., Born, J., (1991): The Parallel Genetic Algorithm as Function Optimizer. In Belew, RK et Booker, LB, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, pp. 271-278.
- Naudts, B., Kallel, L., (1998): *Some facts about so called GA-hardness measures*. Technical report 379 at the CMAP, Ecole Polyt., France.
- Naudts, B., Kallel, L., (2000): Comparison of summary statistics of fitness landscapes. *IEEE Transaction on Evolutionary Computation*.
- Oppacher, F., Wienberg, M. (2000): Reconstructing the Shifting Balance Theory in a GA: Taking Sewall Wright Seriously. *Congress of Evolutionary Computation*.
- Păun, G., (2000): Computing with membranes, *Journal of Computer and System Sciences*, 61(1) 108-143.
- Pelikan, M., Goldberg, D.E., Lobo, F., (1999): *A survey of optimization by building and using probabilistic models*. Technical report IlliGAL 99018, Univ. of Illinois.
- Perrier, J. Y., Sipper, M., Zahnd, J., (1996). Toward a Viable, Self-Reproducing Universal Computer. *Physica D* 97, 335-352.
- Platt, J., (1998): Fast training of support vector machines using sequential minimal optimization. In: *Advances in Kernel Methods – Support Vector Learning*. Cambridge, MA:MIT Press.
- Potter, M.A. (1992). A genetic Cascade-Correlation learning algorithm. In *Proceedings of COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*, IEEE Computer Society Press. pp. 122-133.
- Potter, M.A., De Jong, K.A. (1994): A Cooperative Coevolutionary Approach to Function Optimization. In: Davidor, Y. and Schwefel, H.P., (Eds.), *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, Germany 249-257.

- Potter, M.A., De Jong, K.A. (2000): A Cooperative co-evolution: an architecture for evolving co-adapted subcomponents. *Evolutionary Computations*, 8(1). 1-20.
- Pudil, P., Novovicova, J., Kittler, J., (1994): Floating search methods in feature selection. *Pattern Recognition Letters* 15, 1119-1125.
- Quinlan, J.R., (1993): *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publisher.
- Rasmussen, S., Knudsen, C., Feldberg, R., (1992): Dynamics of Programmable Matter. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pp 159-210, Redwood City, CA, Addison-Wesley.
- Rasmussen, S., Knudsen, C., Feldberg, R., Hindsholm, M., (1990): The Coreworld: Emergence and evolution of cooperative structures in a computational chemistry. *Physica D*, 42, 111.
- Ray, T. S., (1992): An approach to the synthesis of life. *Artificial Life II*, Langton C. G. et al. (Eds.), Addison-Wesley, 371-401.
- Reich Y. (1994): Layered Models of Research Methodologies. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 8(4):263-274.
- Roubos, J.A., Setnes, M., Abonyi, J., (2003): Learning fuzzy classification rules from labeled data. *International Journal of Information Sciences*, (150) 77-93.
- Rubinstein, R. Y., (1999): The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1(2):127-190.
- Sayama, H. (1998): Introduction of structural dissolution into Langton's self-reproducing loop. In C. Adami, R. K. Belew, H. Kitano, C. E. Taylor (Eds.), *Artificial Life VI: Proceedings of the Sixth International Conference on Artificial Life*. Cambridge, MA: MIT Press. (pp. 114-122).
- Sayama, H. (1999): A new structurally dissolvable self-reproducing loop evolving in a simple cellular automata space. *Artificial Life* 5, 343-365.
- Schlierkamp-Voosen, D., Muhlenbein, H., (1994): Strategy adaptation by competing subpopulations. In: Davidor, Y. and Schwefel, H.P., (Eds.),

- Proceedings of the Third Conf. on Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, Germany 199-208.
- Schnecke, V., Vornberger, O., (1996): An adaptive parallel genetic algorithm for VLSI-Layout optimization. *Proceedings of the Fourth Int. Conf. on Parallel Problem Solving from Nature*, Springer-Verlag, Berlin, Germany 859-868.
- Schoenauer, M., Xanthakis S., (1993): Constrained GA optimization. *Proceedings of the Fifth International Conference on Genetic Algorithms*.
- Schwefel, H.P. (1995): *Evolution and Optimum Seeking*. John Wiley and Sons, New York.
- Sipper, M., (1994): Non-uniform cellular automata: Evolution in rule space and formation of complex structures, *Artificial Life IV*, The MIT Press, Cambridge, Mass., 394-399.
- Sipper, M., (1995): Quasi-uniform computation-universal cellular automata, *Lecture Notes in Computer Science*, Springer-Verlag, vol. 929, 544-554.
- Sipper, M., (1996): Co-evolving non-uniform cellular automata to perform computations, *Physica D*, vol. 92, 193-208.
- Sipper, M., Rupp, E. (1996): Co-evolving cellular architectures by cellular programming. In: *Proceedings of IEEE 3-rd Int. Conference on EC*, 306-311.
- Smith, A., Turney, P., Ewaschuk, R. (2003): Self-replicating machines in continuous space with virtual physics. *Artificial Life*, 9(2), 21-40.
- Smith, S.F., (1983): Flexible learning of problem solving heuristics through adaptive search. *Proceedings of the 8th International Joint Conference on AI*.
- Spector, L., (2001): Autoconstructive evolution: Push, pushgp, and pushpop. *Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann Publishers.
- Stanley, K.O., Miikkulainen, R., (2002): Efficient Reinforcement Learning through Evolving Neural Network Topologies. *Proceedings of the Genetic and Evolutionary Computation Conference*, New York, 569-578.
- Stanley, K.O., Miikkulainen, R., (2003): A Taxonomy for Artificial Embryogeny. *Artificial Life*, 9(2), 93-130.

- Szathmáry, E., Maynard Smith, J., (1995a): The major evolutionary transitions. *Nature* 374, 227-232.
- Szathmáry, E., (1995b): A classification of replicators and lambda-calculus models of biological organization. *Proc. R. Soc. Lond. B* 260, 279-286.
- Sohier, C., Denis, B., Bourdet, P., (1996): Applying eco problem solving to the control of an adaptative manufacturing cell. *IMACS-IEEE Multiconference on Computational Engineering in Systems Applications*.
- Tempesti, G., (1995): A new self-reproducing cellular automaton capable of construction and computation. In F. Moran, A. Moreno, J. J. Merelo, & P. Chacón (Eds.), *Third European Conference on A. Life*, LNCS929. Heidelberg, Germany: Springer. 555-563.
- Tulai, A., Oppacher, F., (2002a): Combining Competitive and Cooperative Coevolution for Training Cascade Neural Networks. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, New York. Morgan Kaufmann Publishers, 610-618.
- Tulai, A., Oppacher, F., (2002b): Parallel Genetic Algorithms with Strategy Parameters Encoded as Chromosomes. *Proceedings of the LASTED International Conference on Artificial and Computational Intelligence*, pp. 84-89.
- Tulai, A., (2003): *Emergence and Evolution of Computational Habitats*. PhD thesis proposal. Carleton University, Ottawa, Canada.
- Tulai, A., Oppacher, F., (2004a): The role of speciation in classification tasks. *Proceedings of the LASTED International Conference on Artificial Intelligence and Applications*, pp. 103-108.
- Tulai, A., Oppacher, F., (2004b): Multiple Species Universal Voting: an Evolutionary Algorithm for Supervised Learning. *Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*, pp. 118-125.
- Tulai, A., Oppacher, F., (2004c): Multiple Species Weighted Voting: A Genetics-Based Machine Learning System. *Proceeding of the Genetic and Evolutionary Computation Conference*, New York, Morgan Kaufmann Publishers.

- Tulai, A., Oppacher, F., (2004d): Maintaining Diversity and Increasing the Accuracy of Classification Rules through Automatic Speciation. *Proceedings of the Congress on Evolutionary Computation*, Portland, Oregon, USA.
- Turing, A., (1950): Computing machinery and intelligence. Available on-line. *Mind*, v.59(236), 433-460.
- Turing, A., (1952): The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society B*, 237, 37-72.
- Varela, F. J., Maturana, H. R., Uribe, R. (1974): Autopoiesis: The Organization of Living Systems, its Characterization and a Model, *BioSystems* 5, 187-196.
- Valiant, L. G., (1984): A theory of the learnable. *Communications of the Association of Computing Machinery*, 27(11):1134-1142.
- Vapnik, V. N., (1998): *Statistical Learning Theory*. Wiley, New York, NY.
- Vassilev, V. K., Fogarty, T. C., Miller, J. F., (2000): Information Characteristics and the Structure of Landscapes. *Evolutionary Computation* 8(1): 31-60.
- Voigt, H.M., (1995): Soft genetic operators in evolutionary algorithms. In W. Banzhaf and P.H. Backman, (Eds.). *Evolution and Biocomputation*, Springer-Verlag, Berlin New York.
- von Neumann, J., (1966): *Theory of self-reproducing automata*. Edited and completed by A.W.Burks. Urbana, IL: University of Illinois Press.
- Wagner, K., (1999): Habitat, communication and cooperative strategies. *Proceedings of the Genetic and Evolutionary Computation Conference* 694-701.
- Wah, B.W. and Qian, M. (2000): Constrained Formulations for Neural Networks Training and Their Applications to Solve the Two-Spiral Problem. *Proceedings of the Fifth International Conference on Computer Science and Informatics*.
- White, T., Oppacher, F., (1994): Adaptive Crossover Using Automata. In Davidor, Y. and Schwefel, H.P., (Eds.), *Proceedings of the 3rd Conference on Parallel problem Solving from Nature*, Springer-Verlag, Berlin, Germany 229-238.
- Whiteson, S., Stone, P., (2003): Concurrent layered learning. *Second International Conference on Autonomous Agents and Multi-agent Systems*.

- Wilson, S.W., (1995): Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2) 149-175.
- Wilson, S.W., (1998): Generalization in the XCS classifier system. In: *Genetic Programming Proceedings of the Third Annual Conference*, J. Koza et al., eds., San Francisco, CA: Morgan Kaufmann 665-674.
- Winkeler, J. F., Manjunath, B. S. (1998): Incremental Evolution in Genetic Programming, *Proceedings of the Third Annual Conference on Genetic Programming*, 403-411.
- Witten, I.H., Frank, E., (2000): *Data mining practical Machine Learning tools and techniques with Java implementations*. Morgan Kauffman Publishers.
- Wolfram, S. (2002): *Cellular Automata and Complexity*. Westview press, 2nd edition.
- Yao, X. (1999): Evolving Artificial Neural Networks. *Proceedings of the IEEE* 87(9):1423-1447.
- Yoshii, S., Ohashi, S., Kakazu, Y., (1998a): Self-organized Complexity in Computer Program Ecosystem, *Proceedings of the Sixth Int. Conf. on Artificial Life* C. Adami, R. K. Belew, H. Kitano, and C. E. Taylor, editors MIT Press.
- Yoshii, S., Ohashi, S., Kakazu, Y., (1998b): Modeling of Emergent Ecology for simulating adaptive behavior of universal computer programs. In *From Animals to Animats 5*. R. K. Belew, H. Kitano, and C. E. Taylor, Editors MIT Press.
- Zlochin, M., Birattari, M., Meuleau, N., Dorigo, M., (2001): *Model-based search for combinatorial optimization*. Technical Report TR/IRIDIA/ 2001-15, IRIDIA, Université Libre de Bruxell.

APPENDIX

Tables A.1 – A.8

Table A.1: The average values of the alleles calculated at the end of simulation (the energy conserving case).

Agent	Process	Value
System	txSignal	0.1918108
	rxSignal	0.4874105
	txSignalUpTransducer	0.2247339
	rxSignalUpTransducer	0.3428028
	runReproductiveAgent	0.3164965
	runFeederAgent	0.7239555
	apoptosis	0.0017926
Reproductive	txSignal	0.1086736
	rxSignal	0.3734039
	txSignalUpTransducer	0.4384271
	rxSignalUpTransducer	0.4613666
	produceClone	0.3979564
Feeder development	apoptosis	0.0011168
	energyDistributionGenesis	0.645377
Feeder	txSignal	0.1546911
	rxSignal	0.6100645
	txSignalUpTransducer	0.7041178
	rxSignalUpTransducer	0.6067595
	energyDistribution	0.6373464
	apoptosis	0.0010335
Distributor components	systemDistributor	0.4548534
	reproductiveDistributor	0.2302212
	feederDistributor	0.4319605

Table A.2 : The average values of the alleles calculated at the end of simulation with food replenished every 50 cycles.

Agent	Process	Value
System	txSignal	0.0904144
	rxSignal	0.4941741
	txSignalUpTransducer	0.2603054
	rxSignalUpTransducer	0.5589227
	runReproductiveAgent	0.2727078
	runFeederAgent	0.6835435
	apoptosis	0.0010033
Reproductive	txSignal	0.1763930
	rxSignal	0.5076660
	txSignalUpTransducer	0.3755872
	rxSignalUpTransducer	0.4346239
	produceClone	0.4959855
	apoptosis	0.0014555
Feeder development	energyDistributionGenesis	0.6476648
Feeder	txSignal	0.5143417
	rxSignal	0.5793583
	txSignalUpTransducer	0.6510633
	rxSignalUpTransducer	0.5461579
	energyDistribution	0.6972524
	apoptosis	0.0010254
	systemDistributor	0.3367127
Distributor components	reproductiveDistributor	0.4372435
	feederDistributor	0.2555236

Table A3: Experiment 1: summary of agent mutations after 20000 cycles.

N	Function	Gene	Zone							
			1	2	3	4	5	6	7	8
0	P+	S.apoptosis								
1	S+	S.createOrgans								
2	P+	S.growFeeder								
3	P+	S.growWomb								
4	P+	S.growSignalingSpace								
5	S+	S.createTxRxProcesses								
6	S+	S.createIntakeFoodProcess								
7	S+	S.createTxRxUpTransducerProc								
8	S+	S.createTxRxUpTransducerToPa								
9	P+	S.runFeeder								
10	P+	S.runWomb					2			
11	P+	S.txSignal								
12	P+	S.rxSignal		2						
13	P+	S.txSignalUpTransducer								
14	P-	S.rxSignalUpTransducer								
15	P+	S.rxFoodFromEnvironment								
16	P+	S.txSignalToParent								
17	P+	S.rxSignalFromParent								
18	S+	S.createSystemDistributor								
19	P+	S.systemDistributor								
20	S+	S.wombEnergyDistribCreatorCr								
21	P+	R.apoptosis								
22	P+	R.growSignalingSpace								
23	S+	R.createTxRxProcesses								
24	S+	R.createTxRxUpTransducerProc								
25	P+	R.reproductiveDistributor						2		
26	S+	R.createProduceClone								
27	P+	R.produceClone								
28	P+	R.developOffspring				2				
29	P0	R.txSignal								2
30	P+	R.rxSignal								
31	P+	R.txEnergyUpTransducer								
32	P+	R.txSignalUpTransducer								
33	P+	R.rxSignalUpTransducer				2				
34	P+	R.rxEnergyUpTransducer								

35	S+	F.wombEnergyDistributionCrea		
36	P+	F.apoptosis		
37	P+	F.growSignalingSpace	2	2
38	S-	F.createTxRxProcesses		
39	S+	F.createRxFood		
40	S+	F.createTxRxUpTransducerProc		
41	S+	F.createEnergyDistribution		
42	P+	F.feederDistributor		
43	P+	F.foodProcessing		
44	P0	F.txSignal		
45	P0	F.rxSignal		
46	P+	F.rxFood		
47	P+	F.txSignalUpTransducer		
48	P+	F.rxEnergyUpTransducer	-	-
49	P+	F.txEnergyUpTransducer	d d	
50	P+	F.txSignalUpTransducer		2
51	P-	F.rxFoodUpTransducer		

Table A4: Experiment 1: summary of agent mutations after 20000 cycles.

N	Function	Gene	Zone					
			1	2	3	4	5	6
0	P+	S.apoptosis						
1	S+	S.createOrgans						
2	P+	S.growFeeder						
3	P+	S.growWomb						
4	P+	S.growSignalingSpace						
5	S+	S.createTxRxProcesses						
6	S+	S.createIntakeFoodProcess						
7	S+	S.createTxRxUpTransducerProces						
8	S+	S.createTxRxUpTransducerToPare						
9	P+	S.runFeeder						
10	P+	S.runWomb						
11	P+	S.txSignal						
12	P+	S.rxSignal						
13	P+	S.txSignalUpTransducer						
14	P-	S.rxSignalUpTransducer						

15	P+	S.rxFoodFromEnvironment		
16	P+	S.txSignalToParent		
17	P+	S.rxSignalFromParent		
18	S+	S.createSystemDistributor		2
19	P+	S.systemDistributor		
20	S+	S.wombEnergyDistribCreatorCrea		
21	P+	R.apoptosis		
22	P+	R.growSignalingSpace		
23	S+	R.createTxRxProcesses		
24	S+	R.createTxRxUpTransducerProces		
25	P+	R.reproductiveDistributor		
26	S+	R.createProduceClone		
27	P+	R.produceClone		
28	P+	R.developOffspring		2
29	P0	R.txSignal	-	2
30	P+	R.rxSignal		
31	P+	R.txEnergyUpTransducer		
32	P+	R.txSignalUpTransducer		
33	P+	R.rxSignalUpTransducer		2
34	P+	R.rxEnergyUpTransducer		
35	S+	R.wombEnergyDistributionCreato		2
36	P+	F.apoptosis		
37	P+	F.growSignalingSpace	2	2
38	S-	F.createTxRxProcesses		
39	S+	F.createRxFood		
40	S+	F.createTxRxUpTransducerProces		
41	S+	F.createEnergyDistribution		
42	P+	F.feederDistributor		
43	P+	F.foodProcessing		
44	P0	F.txSignal		
45	P0	F.rxSignal		
46	P+	F.rxFood		
47	P+	F.txSignalUpTransducer		
48	P+	F.rxEnergyUpTransducer	-	
49	P+	F.txEnergyUpTransducer		2
50	P+	F.rxSignalUpTransducer		2
51	P-	F.rxFoodUpTransducer		

Table A5: Experiment 1: summary of agent mutations after 20000 cycles.

No	F	Gene	1	2	3	4	5	6	7	8
0	P+	S.apoptosis								
1	S+	S.createOrgans								2
2	P+	S.growFeeder								
3	P+	S.growWomb								
4	P+	S.growSignalingSpace								
5	S+	S.createTxRxProcesses			2					
6	S+	S.createIntakeFoodProcess								
7	S+	S.createTxRxUpTransducerProcesses								
8	S+	S.createTxRxUpTransducerToParent								
9	P+	S.runFeeder								
10	P+	S.runWomb								
11	P+	S.txSignal								
12	P+	S.rxSignal								
13	P+	S.txSignalUpTransducer								
14	P-	S.rxSignalUpTransducer		-						-
15	P+	S.rxFoodFromEnvironment								
16	P+	S.txSignalToParent								
17	P+	S.rxSignalFromParent		2						
18	S+	S.createSystemDistributor								
19	P+	S.systemDistributor								
20	S+	S.wombEnergyDistribCreatorCreator								2
21	P+	R.apoptosis								
22	P+	R.growSignalingSpace								
23	S+	R.createTxRxProcesses								
24	S+	R.createTxRxUpTransducerProcesses								
25	P+	R.reproductiveDistributor								
26	S+	R.createProduceClone								2
27	P+	R.produceClone		2						
28	P+	R.developOffspring								
29	P0	R.txSignal								
30	P+	R.rxSignal								
31	P+	R.txEnergyUpTransducer		-	-	-	-	-	-	-
32	P+	R.txSignalUpTransducer								2
33	P+	R.rxSignalUpTransducer								
34	P+	R.rxEnergyUpTransducer								

Table A7: Experiment 5: summary of agent mutations after 20000 cycles.

No	P	Gene	Zone									
			1	2	3	4	5	6	7	8	9	0
0	P+	S.apoptosis										
1	S+	S.createOrgans		2	2			2		2		
2	P+	S.growFeeder										
3	P+	S.growWomb										
4	P+	S.growSignalingSpace										
5	S+	S.createTxRxProcesses									2	2
6	S+	S.createIntakeFoodProcess							2	2	2	
7	S+	S.createTxRxUpTransducerProcesses										
8	S+	S.createTxRxUpTransducerToParent							2	2	2	
9	P+	S.runFeeder										
10	P+	S.runWomb										
11	P+	S.txSignal										
12	P+	S.rxSignal			2							
13	P+	S.txSignalUpTransducer										
14	P-	S.rxSignalUpTransducer									2	
15	P+	S.rxFoodFromEnvironment										
16	P+	S.txSignalToParent										
17	P+	S.rxSignalFromParent										
18	S+	S.createSystemDistributor									2	
19	P+	S.systemDistributor										
20	S+	S.wombEnergyDistribCreatorCreator										
21	P+	R.apoptosis										
22	P+	R.growSignalingSpace										
23	S+	R.createTxRxProcesses										2
24	S+	R.createTxRxUpTransducerProcesses										
25	P+	R.reproductiveDistributor						2	2	2	3	
26	S+	R.createProduceClone										
27	P+	R.produceClone		2			2					
28	P+	R.developOffspring										
29	P0	R.txSignal										
30	P+	R.rxSignal										
31	P+	R.txEnergyUpTransducer										
32	P+	R.txSignalUpTransducer										
33	P+	R.rxSignalUpTransducer										
34	P+	R.rxEnergyUpTransducer										

14	P-	S.rxSignalUpTransducer		
15	P+	S.rxFoodFromEnvironment	2	
16	P+	S.txSignalToParent		
17	P+	S.rxSignalFromParent		
18	S+	S.createSystemDistributor		
19	P+	S.systemDistributor		
20	S+	S.wombEnergyDistribCreatorCreator		
21	P+	R.apoptosis		
22	P+	R.growSignalingSpace		
23	S+	R.createTxRxProcesses		
24	S+	R.createTxRxUpTransducerProcesses		
25	P+	R.reproductiveDistributor	2	
26	S+	R.createProduceClone		
27	P+	R.produceClone		
28	P+	R.developOffspring		
29	P0	R.txSignal		
30	P+	R.rxSignal		
31	P+	R.txEnergyUpTransducer	- - -	
32	P+	R.txSignalUpTransducer		
33	P+	R.rxSignalUpTransducer		
34	P+	R.rxEnergyUpTransducer		
35	S+	R.wombEnergyDistributionCreator		
36	P+	F.apoptosis		
37	P+	F.growSignalingSpace		
38	S-	F.createTxRxProcesses	2	
39	S+	F.createRxFood		
40	S+	F.createTxRxUpTransducerProcesses	3	
41	S+	F.createEnergyDistribution		
42	P+	F.feederDistributor		
43	P+	F.foodProcessing		
44	P0	F.txSignal		
45	P0	F.rxSignal	- - -	
46	P+	F.rxFood		
47	P+	F.txSignalUpTransducer		
48	P+	F.rxEnergyUpTransducer	12 27 15	7
49	P+	F.txEnergyUpTransducer		d d d d d
50	P-	F.rxSignalUpTransducer	5 2	-
51	P+	F.rxFoodUpTransducer		

Implementation

The subject of evolving methods and their encoding is tightly coupled to implementation.

The MSWV system as well as the first *SH* implementations presented in Chapter 5 and Chapter 6 has been based on RePast, a software framework for creating agent based simulations using the Java language. RePast has been developed at the University of Chicago Social Science Research Computing department.

Dynamic method definition, dynamic code modification and execution are not a trivial thing in Java. It is possible that other OO languages, for example SmallTalk offer a better environment for *CH* future implementations.

INDEX

A

ACO. *See* ant colony optimization
 ant colony optimization, 27
 Ants Colonies, i, 26 29
 Artificial Chemistries, 19 24
 artificial embryogenesis, 8, 81, 82, 134, 169,
 170, 184
 Avida, 25

C

canalization, 81
 CCC-GA, 5, 6, 36 45, 45, 48, 51, 52, 186
 Cellular Automata, i, 11 17
 CH theoretical framework, 7, 83 104
 CHAM. *See* Chemical Abstract Machine
 Chemical Abstract Machine, 7, 20, 24
 Computational Ecologies, 10
 Computational Habitat, i, 1, 3, 4, 6, 7, 10,
 12, 15, 17, 18, 19, 23, 25, 27, 28,
 31, 32, 33, 36, 43, 83, 87, 99, 172,
 175, 183, 186, 187
 computer programs worlds, 24 26
 Coreworld, 24
 cross-entropy, 27

D

DNA computing, 15, 23

E

Eco-problem solving, 34
 EDA. *See* estimation of distribution
 algorithms
 estimation of distribution algorithms, 27
 Evolutionary Strategies, 2, 41
 external signaling space, 88, 89, 140

F

flooding, 22, 124

G

GBML, i, 7, 53, 62, 78, 187
 generalized CCC-GA, 46, 52, 187
 Genetic Algorithms, i, 2, 3, 23, 26, 36, 45,
 186
 genetic operators, 4, 30, 45, 46, 47, 50, 53,
 164, 165, 184, 200
 genotype, 4, 8, 11, 14, 16, 25, 30, 55, 81, 91,
 92, 96, 134, 135
 Gompertz, i, 143 50
 graded sequences of problems, 4

H

habitat signaling space, 88, 113, 115, 118
 heuristic search algorithms, 26

I

incremental evolution, 5
 Incremental Evolution, 29 33
 internal signaling space, 88, 89, 112, 121,
 139
 ISS. *See* internal signaling space

L

Langton's loop, 12

M

membrane computing, 23
 model-based search, 26
 MSUV, xii, 172, 174
 MSWV, i, xii, 7, 53, 172, 174
 MSWV algorithm, 52 82
 multi-agent systems, 33 35

N

natural computing, 23

networks of problems, 3
networks of solutions, 3
Non-Uniform Cellular Automata, i, 17 19

P

parallel Genetic Algorithms, 40
parameter free algorithms, 6, 78, 186, 187
phenotype, 4, 8, 11, 14, 16, 25, 55, 91, 96,
105, 134, 137, 141, 150, 152, 155,
156, 157, 161, 164, 169, 172, 184
primordial soup, 14, 18, 21, 22
protein space, 2
P-system, 23
Pushpop, 26

Q

quantum computing, 23

R

Reversible Cellular Automata, 14
rich environment, 3

S

self-replication, 11, 14, 15, 17, 19, 21
self-reproducing automaton, 10
SGA. *See* stochastic gradient ascent
signaling, 12, 25, 89
signaling path differentiation, 9, 143, 153,
159, 161, 167
signaling space, 87, 106, 107, 138, 141
signaling space distance, 141

speciation, 7, 36, 37, 52, 57, 59, 69 78, 69
78, 173, 174, 176, 178, 181, 183,
186

stochastic gradient ascent, 27

Stochastic Habitat, 8, 83, 99, 134, 168, 187

Stochastic Habitats, 150

strategy parameters, 6, 36, 45 52, 186, 187

structurally dissolvable self-reproducing
loop, 15

T

Tierra, 24

transcription, 11

translation, 11

Turing, 10, 184

U

unbounded evolutionary capability, 11, 114,
183, 184

universal computational capability, 18, 24

universal construction capability, 11

V

von Neumann, 10

W

worker stochastic process, 89

X

XSS. *See* external signaling space