Learning Classifier Systems for Multi-objective Robot Control

Matthew Studley

Learning Classifier Systems Group Technical Report – UWELCSG06-005

Faculty of Computing, Engineering and Mathematics

University of the West of England
Coldharbour Lane, Frenchay
Bristol BS16 1QY, UK.

Abstract

The work presented in this Thesis concentrates on the practicalities of using Learning Classifier Systems (LCS) for control problems in which there exists more than one goal. The emphasis is always upon the use of LCS on physical robot hardware.

A number of simple simulated problems with multiple goals are introduced. Two classifier systems, ZCS and XCS, are applied to these problems. It is found that ZCS can solve the simplest problems given suitable parameterisation, and the performance landscape is explored with respect to these parameter settings. However, ZCS is not shown to be able to solve slightly more complex problems. XCS is shown to be capable of solving these more complex problems with little need to adjust parameters, and it is shown to be suitable to use in the physical world by replacing the usual random 'explore' policy.

TCS, an extension of ZCS, is then applied to similar problems in the physical world. TCS cannot be shown to be capable of solving these problems, presumably due to the same parameterisation problems. X-TCS is presented; an accuracy-based classifier system working in continuous space on a hardware platform. X-TCS solves problems of robot control with multiple goals optimally, quickly, and with little need for parameter adjustment. X-TCS, which extends TCS, may be of interest to the Reinforcement Learning community due to its in-built ability to discover appropriate levels of discretization in the problem space, requiring no *a priori* discretization or additional techniques to be used.

Since this work is focused on continuous learning, rather than 'learn' then 'perform', X-TCS may be suited to control in non-stationary conditions. An example might be to maximise energy efficiency in an engine management system given different grades of fuel, and despite ongoing changes in performance characteristics due to mechanical degradation.

Acknowledgements

I would like to take this opportunity to thank my supervisor, Professor Larry Bull, for his unfailing interest, guidance and wisdom throughout this project. Similarly, I owe a huge debt of gratitude to Dr Jacob Hurst for his help and friendship.

In addition, I would like to thank the members of the UWE Learning Classifier Systems group and in particular, Dr Alwyn Barry and Dr Tony Pipe, for their support, questions and contributions.

My heartfelt thanks also go to the members of the UWE IAS Lab. Professor Chris Melhuish has been both a source of practical advice, a sounding-board, and a witty and erudite friend, and the other members of the Lab have helped enormously to extend my understanding and expand my horizons in the field of robotics.

Finally, none of the robot experiments would have taken place had it not been for the support staff of the IAS Lab. Moving from the safely-constrained world of simulation to physical robotics involves a massive increase in complexity, where the casual assumptions of determinism upon which experimentation and fault-finding are based '*in virtua'* provide surprising traps for the unwary and naïve. Ian Horsfield and Chris Bytheway have helped me realise that sometimes all that is necessary is some extra grease or a tightened nut!

I have enjoyed the last three years enormously! Thank you all for your support, friendship and conviviality.

Contents

Learning is but an adjunct to ourself,
It adds a precious seeing to the eye.

Shakespeare, “Love’s Labours Lost”, Act 4, Scene 3.

Chapter 1 Introduction

Mankind has long recognised the desirability of autonomous artefacts which can perform work on our behalf. Literature and legend are filled with references to the likes of the man of brass, Talos, which guarded Crete and was destroyed by the Argonauts [Apollonius], Rabbi Loew’s Golem which was created to free the Jews of Prague from drudgery [Thieberger1955], Rossum’s Universal Robots; the slaves of a new world [Capek1923], and the Monster of Mary Shelley’s *Frankenstein* [Shelley1818].

Ignoring the (sometimes severe) complications that dawning consciousness brings to these fictional creations, we are still impelled by the lure of creating artificial intelligence today. Although robots feature in much modern fiction and have found some static niches within industrial settings on carefully constrained problems, they are not yet widely available in a larger context. One reason for this is a lack of adaptability or learning. To deal with complex environments and changing conditions, robots must be capable of adapting their actions in order to be useful. This motivates the work presented here; the focus is on continuous learning, rather than a ‘training phase’ followed by performance in the absence of further learning.

All living organisms exhibit the properties of homeostasis. While their external environment changes constantly, they are able to maintain a relatively constant internal environment - when their ability to do so is compromised, death may swiftly follow. In order to maintain this stability, they must balance many conflicting goals or desires, and dynamically change their actions according to their circumstances. The needs of the moment may over-ride longer-term objectives; the need for food is less than the need to avoid being eaten, the need for shelter may overcome the drive to mate, etc. Balancing these conflicting drives is the difference between success and failure, between life and death, and the optimization of balancing behaviours is therefore subject to great evolutionary pressure.

For any autonomous artificial entity to perform useful work, it will need to operate in complex problem domains, and also will need to adapt its behaviour to optimize the balance of tasks it must perform. This thesis considers the use of an adaptive system – here a Learning Classifier System (LCS) [Holland1976] – to control an agent that has to solve multi-objective problems. For example, in a simple case, the agent has to seek the shortest path to a goal state while it also maintains its energy levels, in similar fashion to a mobile robot that must perform tasks while maintaining its battery power. The optimal course of action can change dynamically during the course of performing a trial for its energy levels may be depleted by its actions; the learner is operating in a changing environment.

Robotic platforms are a useful test bed for Artificial Intelligence (AI). In the real world, decisions must be timely, data is noisy, movements are imperfect and processing power is constrained to preserve battery life. Testing AI theories on machines that interact with the real world through physical sensors and effectors reveals areas of weakness. New theories have been born from the inadequacies in a robotic context of systems that worked well in a disembodied setting. Finally, autonomous physically situated entities are desirable; like the Golem they can do repetitive tasks that humans either do not wish to perform, or are unable to perform. If we want to create physically situated autonomous entities we cannot expect to solve the inherent problems in a virtual environment, as shall be discussed below.

The work presented in this thesis concentrates on one important element of adaptability for autonomous physical robots. We demonstrate that multiple behaviours may be optimally learnt using Learning Classifier Systems, as may the co-ordination of these behaviours such that a learner may switch between the behaviours in a fashion that is itself optimal. This may be necessitated in circumstances where the robot has multiple (conflicting) goals that must be balanced.

In this chapter, we briefly explore some of the history of AI. We then examine two approaches inspired by the study of adaptation in biology, namely Evolutionary Computing (EC) and Reinforcement Learning (RL). We then introduce Learning Classifier Systems, which combine both of these approaches, and explain why LCSs may be more suited to the task of robot control than either EC or RL in isolation. Where possible, we mention how such approaches have been used for problems with multiple objectives.

1.1 Artificial Intelligence

In seeking to explain the behaviours of men and animals alike, an influential school of thought arose in the early part of the 20th Century. Behaviourism, based upon the philosophical tradition of the British Empiricists such as Locke, asserted that behaviour should not be studied by the attribution of assumed mental states or processes. The external behaviour of a person is not accounted for by referring to the internal behaviour of the person (say, his or her internal problem solving or thinking) if, therein, the behaviour of the person is unexplained. '*The objection*', wrote Skinner, '*to inner states is not that they do not exist, but that they are not relevant in a functional analysis*' [Skinner1953]. 'Not relevant' means, for Skinner, explanatorily circular or regressive. (There are obvious flaws with the behaviourist stance; it disregards innate behaviours and capacities, and fails to explain the human condition

– the ‘qualia’ of e.g. pain is *experienced*, in addition to the generation of appropriate pain-reduction behaviour which is observed from outside.)

The early pioneers of computing such as Alan Turing formulated the belief that a ‘Universal Machine’ which could model any abstract computing engine could therefore simulate human intelligence. The ‘Church - Turing’ thesis[1] suggests that a problem which cannot be solved by such a theoretical machine cannot be solved by a human mind, and conversely that any problem soluble by a human is equally soluble by the machine.

Since the Dartmouth Conference in 1956, AI became thought of as generating abstract intelligences that could solve all problems, providing that the environment of the intelligence can be suitably presented to them. Based upon this doctrine of functional equivalence (for which the empiricism of the behaviourists is surely a *sine qua non*, stripping away mentalist models and innate drives), and the concept that intelligence is synonymous with computation, the position of classical AI is represented by the ‘Physical Symbol Systems Hypothesis’ (PSSH) of Newell and Simon [Newell1976]. This asserted that ‘*a physical symbol system has the necessary and sufficient means for general intelligent action*.’ Physical symbols have some *physical existence*, whether as marks on paper, electronic charges in computer registers, or in some fashion in the physical brain that instantiates a mind. The PSSH asserts that physical symbols are *necessary* for intelligence, that systems based upon them are *sufficient* to provide for intelligence with no addition, and such systems are widely and *generally applicable*. The difference between Machine and Human Intelligence is one of substrate and plumbing; instantiation.

[1] The term 'Church-Turing thesis' seems to have been first introduced by Kleene:
'So Turing's and Church's theses are equivalent. We shall usually refer to ... versions which deal(s) with 'Turing machines' as the Church-Turing thesis.' [Kleene1967]

Harnad presents a good overview of the characteristics of symbol systems [Harnad1990], quoted below.

'A symbol system is:

1. *a set of arbitrary 'physical tokens' scratches on paper, holes on a tape, events in a digital computer, etc. that are*
2. *manipulated on the basis of 'explicit rules' that are*
3. *likewise physical tokens and strings of tokens. The rule-governed symbol-token manipulation is based*
4. *purely on the shape of the symbol tokens (not their 'meaning'), i.e., it is purely syntactic, and consists of*
5. *'rulefully combining' and recombining symbol tokens. There are*
6. *primitive atomic symbol tokens and*
7. *composite symbol-token strings. The entire system and all its parts -- the atomic tokens, the composite tokens, the syntactic manipulations both actual and possible and the rules -- are all*
8. *'semantically interpretable' The syntax can be systematically assigned a meaning e.g., as standing for objects, as describing states of affairs.'*

Intelligence is therefore reduced to the application of operators to data. Given some sensory input, there exists a set of transformations which embody logical thought, and which thereby produce a set of outputs which we would recognise as being the product of intelligence in that they enable the successful attainment of a goal state. This is the position which has now become known as 'classical AI'.

Newell and Simon implemented the theories formalised in the Physical Symbol Hypothesis in a system they termed the General Problem Solver (GPS), and described originally in 1957

[Newell1957]. GPS was an example of a *production system* – symbols are manipulated according to rules, and limited inferences can be drawn. Production systems of this type became known as *expert systems*. For example, MYCIN [Shortliffe1973] has many rules about how to infer which bacterium is causing an illness based on symptoms and the result of laboratory tests. However, its formalism has no way of expressing the fact that bacteria are organisms that grow within the body.

It is important to note that the symbols manipulated in such a system are operated upon on the basis of their ‘shape’; processing is on syntax, rather than semantics. Such a system is therefore ‘ungrounded’; the symbols being processed have no meaning within the confines of the system, although they can be assigned a meaning from outside the system. MYCIN has no knowledge that bacteria are physical organisms growing in the body of the patient or in the culture vessel. This is exemplified by Searle’s famous ‘Chinese Room’ argument [Searle1980], and its derivatives. Although the rules within the system allow for the manipulation of symbols, since the symbols have no meaning, no component of the system can be said to be intelligent, although to the intelligent observers outside the system its behaviour might appear intelligent. This is unlike the intelligence of a biological mind in which meaning must be intrinsic, and hence symbol manipulation cannot provide the ‘***sufficient*** *means for* ***general*** *intelligent action*’ claimed by Newell and Simon [ibid., emphasis added]. Harnad provides a further example – consider the task of learning Chinese as a *first* language from a dictionary giving Chinese definitions of Chinese pictograms. There is no way to bootstrap the process; there is no grounded starting point, no ‘Rosetta Stone’ that enables semantic interpretability.

Within constrained problem domains in which the problem of externally attributing meaning to symbol is readily solved, the approach has been successful. Programs have been written to solve a class of problems that give humans intellectual difficulty: examples are playing chess, proving mathematical theorems, transforming one symbolic expression into another by given rules, integrating expressions composed of elementary functions, determining chemical compounds consistent with mass-spectrographic and other data. Game playing has produced notable successes, such that of IBM's 'Deep Blue' series of chess-playing systems against the best human opponents. Classical AI has been similarly useful in other abstract domains, such as the automatic translation of defined and constrained knowledge in database applications and service negotiation where the problem of semantic mark-up has been addressed by humans. KIF[2], the Knowledge Interchange Format is an example of this.

Despite their successes, when classical AI techniques have been applied to robotics they prove difficult to implement and fragile. The frequently cited example of the robot 'Shakey' [Nilsson1984] will demonstrate both the successes and failures of this approach. Shakey was developed at the Stanford Research Institute between 1966 and 1972, and was provided with a specially constructed set of rooms in which to operate. It was equipped with a world model containing representations of the named rooms, doors, and boxes, and could be tasked to enact goal-directed instructions such as moving from one room to another using a set of pre-coded action routines.

In order to determine which actions to take to achieve its goals, Shakey used the 'STRIPS' planning system [Fikes1972]. STRIPS, like the GPS of Newell and Simon, is a hierarchical planning system; to achieve a goal, the problem is broken down into sub-goals. In STRIPS, models of the action routines – which might have preconditions associated with them – allowed the prediction of an action's effects. STRIPS thus searched through the possible

[2] draft proposed American National Standard (dpANS) NCITS.T2/98-004
http://logic.stanford.edu/kif/dpans.html

action sequences that would allow the robot to attain the goal state in order to formulate a plan.

Shakey was slow in computation due to the limitations of the processing platform available at the time. More importantly, it could only operate in the specially constructed environment that matched its internal model. This illustrates the fragility of using a predefined model with no ability to adapt or generalise. More important still, the approach scales poorly. As the number of decision points within the decision hierarchy grows, the time required to search for a solution grows exponentially. A system which has to ‘sit and think’ for hours before acting may find that the world has changed around it making its decisions invalid.

In his seminal 1961 paper, Marvin Minsky [Minsky1961] identified the following sub-problems in building artificial intelligences; Search, Pattern-Recognition, Learning, Planning, and Induction. If pattern-recognition is separable from the other processes comprising an intelligent system, then to ground a symbol-based system is a matter of connecting the ‘intelligence’ to the world in the right way. Unfortunately, the problem of connecting to the world in this way cannot be solved in isolation from cognition[3]. A system is grounded because there is no necessity for an outside intelligence to attribute meaning to the syntactic symbols. Minsky discussed the use of connectionist systems for pattern recognition, though he concluded that their contemporary form as perceptrons [Rosenblatt1958] was of limited use.

[3] In mammals, vision is learnt (upon a prenatally developed infrastructure) through changes in neuronal connections in the visual cortex; a young animal raised in the absence of visual stimuli (due to its eyes being sewn shut) may be unable to see when its eyes are opened, even though the retina is undamaged. See [Hubel1988] for an overview by one of the main pioneers of such investigations. To misapply the epigraph from *‘Love’s Labours Lost’*, ‘…learning…adds a precious seeing to the eye’ since without learning - the adaptive processes of differential neuronal growth and connection - the eye is blind.

In contrast to the symbol manipulating approaches, an alternate approach has arisen which is often termed ‘sub-symbolic’. This is based upon the concept of having many (simple) sub-units with complex interactions – Rosenblatt’s perceptrons were an early example of artificial neural networks in which the sub-units to some degree model the behaviours of biological nerve cells. Such connectionist machines can be labelled ‘sub-symbolic’ since there is no concept of rule-governed combining and recombining of symbols. Connectionist machines are grounded since they are capable of learning to distinguish categories within input data, and assigning new data to those categories. No external intelligence is necessary to assign semantic content to syntactic symbols; semantic information arises within the system itself.

It was in reaction to the problems exemplified by ‘Shakey’ that Rodney Brooks of MIT spearheaded the movement which was to become known as ‘behaviour-based robotics’. Rather than the sequence of Sense-Model-Plan-Act of earlier approaches, Brooks’ model was based upon the principle that behaviour could be composed of modules, each of which received data directly from the environment, and which could directly produce actions via the robot’s effectors. The co-ordination of these behaviours could be realised via a ‘subsumption architecture’, in which higher-level behaviours could over-ride more-frequently triggered lower-level behaviours, the action of the robot being composed of either some combination of the actions proposed by all triggered behavioural modules or only that of the highest level ‘layer of competence’.

One advantage of this approach is that behaviours can be developed in isolation, and once developed re-used, in a manner in some ways analogous to the conservation of structure and information evinced in the evolution of life. Developing robotic systems to perform complex tasks is therefore simplified through the use of pre-defined behavioural components. Also, directly coupling sensors to effectors via behavioural modules rather than having a central planning process using the sensor data against an internal model promotes real-time behaviour; when the world changes, the robot responds. This too has biological justification. For example, the 'knee-jerk' response generated by a reflex arc is a direct link from sensors to effectors, like other reflexive responses thus requiring no 'higher processing'[4] or decision making.

Rather than having the internal model of the classical approaches, Brooks claimed that since '*the world is its own best model'* [Brooks1986] there is no necessity to either supply or build such an internal representation. The intelligence displayed by the system is hidden within the co-ordination of layers of competence, and shown by the responsiveness of the system to its changing environment. This approach was pioneered in the 1950s by W. Grey Walter with his 'Tortoise' robots, built using analogue electronics and mechanical components, and capable of autonomous phototaxis and, perhaps, more complex behaviours. Owen Holland presents a review of this work of Grey Walter in [Holland1997].

As a pragmatic methodology it offers gains in terms of the speed of response to environmental input, and the possibility to reuse components developed in isolation from each other. Brooks also concentrated on embodied systems *in situ* rather than attempting to create general-purpose systems in simulated environments which could then prove difficult or impossible to integrate with complicated, noisy data on physical platforms. However, there are some problems with this sub-symbolic architecture. The co-ordination of behaviours and the mechanisms of subsumption implicitly replace the global model. This co-ordination becomes more complex as more modules are added, and requires careful design and perhaps considerable *ad hoc* adjustment in order that behaviours should be selected in an appropriate fashion. Also, if the goal is to produce the type of intelligence we recognise in each other

[4] Although some reflexes involve intermediary neurons between sensory and motor neurons, these are most commonly in the spinal column of vertebrates, rather than the brain.

then it is difficult to see how this approach can succeed alone; our conscious experience of life suggests that we are not merely driven in our actions by circumstances.

There now follows a brief overview of some areas of research related to the Learning Classifier Systems used in this thesis, including where possible evidence of their application to problems with multiple objectives.

1.2 Evolutionary Computation

Evolution by selective pressure can be simply summarised;

- Individuals in a population differ from each other.
- These difference are to some degree heritable.
- The differences between individuals determine their success in profiting from their environment and in their interactions with other members of the population, and therefore determine the percentage of the next generation comprised of their offspring.

In this way, a population changes in response to the challenges of its environment. The simulation of evolution by selective pressure can thus allow for a population of candidates encoding possible solutions to a problem to converge upon its optimal solution – all that is needed is some measure of the relative fitness of the candidates in solving the problem, and a means by which these differences in fitness alter the replication of the candidates. In order to sample areas of the search space not represented in the population, new candidates must be generated, typically through processes analogous to those of genetic change in natural organisms.

There have been a number of approaches to simulate evolution within computers, including Holland’s Genetic Algorithms (GA) [Holland1975], Rechenberg’s Evolution Strategies (ES)

[Rechenberg1965], Koza's Genetic Programming (GP) [Koza1992], and the Evolutionary Programming of Fogel et al. [Fogel1966]. Since Learning Classifier Systems (LCS) stem from GAs, the latter are briefly considered below.

1.2.1 Genetic Algorithms (GAs)

Within the prototypic GA, a population of chromosomes is initially created at random. These chromosomes are typically binary strings that in some way encode possible solutions to a problem. As there is a mapping between the chromosomal representation and the solutions, the former are considered as the genotypes that explore the *search space*, and the latter as the phenotypes which search the *problem space*.

Each chromosome in the population has an associated *fitness* value. This represents the utility of the phenotype, typically assessed by a *fitness function*. When the *relative fitnesses* of the phenotypes have been assessed, reproduction can take place. Chromosomes have a chance of reproducing proportional to their relative fitness; a variety of schemes are used to implement this, such as *tournament selection* and *roulette-wheel selection*. In reproduction, two parental chromosomes may be *recombined* through *crossover* at one or more randomly-chosen points to produce offspring. There is also a constant probability that each position, or *allele*, in the chromosome of the offspring is changed at random in a process analogous to *mutation*.

Since the number of individual chromosomes in the population is typically fixed, there must be a strategy for replacing old chromosomes with the ones generated by reproduction. In Holland's GA, this process was *generational*, that is, the entire population is replaced by new members on each iteration of the algorithm. Again, other approaches are possible, such as the *steady-state* method [Syswerda1989] in which only a few lower-fitness members of the population are replaced on each iteration. Once the genetic operators have produced new individuals, relative fitness is again assessed and the cycle begins again. The cycle is terminated when either some fitness criteria is reached, for example, a phenotype has perfect absolute fitness, or after some predetermined number of cycles have been performed.

1.2.2 Schema Theory and the Building Block Hypothesis

In order to prove the utility of GAs, Holland presented an application of *schema theory* [Holland1975]. In schema theory, the search space is partitioned into subspaces of varying levels of generality – the schemata - and mathematical models are constructed which estimate how the number of individuals in the population belonging to certain schema can be expected to grow in the next generation. From this model arose the building block hypothesis (BBH) [Goldberg1989], which attempted to explain how a GA solves a problem by positing that near-optimal solutions were forged from small, low-order, fitter-than-average schemata.

A *schema* is a ternary string consisting of symbols from the set {0,1,*}, in which '*' –'don't care' - is a meta-character that matches both 0 and 1, and thereby provides for generalisation. In an alphabet of *k* characters, for a string of length *l* there are k^l different strings. Introducing the meta-character * means there are $(k+1)^l$ schemata. Since there are more schemata than there are (binary) strings, the fitness evaluation of a single string implicitly provides information about the fitness of a greater number of schemata. There is thus an implicit parallelism in the search process.

In general, any particular string is a member of 2^l schemata because each position may take on its actual value, or a don't care symbol. As a result, a population of *n* strings contains somewhere between 2^l (if all the strings are identical) and $n \ x \ 2^l$ (if all the strings are different), schemata - thus even a moderately-sized population contains a wealth of information about important similarities. Holland showed how the proportion of schemata varied in the population according to the relative fitness of (the phenotypes of) the genotypes which contain them.

A schema *H=0***0*** represents all strings where *l=7* and with the specific character 0 at positions one and five. *H* stands for hyperplane. The *order, o(H),* of the schema is the number of specific characters in the schema, here two, and the *defining length, d(H),* is the distance between the outermost specific characters. In this case *d(H)=4,* since the first specific character is at position one and the last at position five.

Let $m(H, t_{+1})$ be the number of schemata *H* found in the population at time t_{+1}. The probability of reproduction is dependent on relative fitness, which can be represented as

$$\frac{f(H)}{f_{av}}$$

where *f(H)* is the mean fitness of individuals containing *H* and f_{av} is the average fitness of the population as a whole.

If crossover occurs between the outermost defining characters of the schema, the schema will be disrupted. With one-point crossover this will happen with the probability *d(H)/(l-1).* Mutation can occur with equal probability at each defining character of the schema, thereby disrupting it with probability proportional to the order of the schema *o(H).*

Thus;

$$m(H,t_{+1}) \geq m(H,t)\cdot\frac{f(H)}{f_{av}}\cdot\left[1-\left(p_c\cdot\frac{d(H)}{l-1}\right)-p_m\cdot o(H)\right]$$

where p_c is the probability of crossover, and p_m is the probability of mutation for each character, or *allele*, of the chromosome.

This is an inequality since there may also be recruitment due to the creation of the schema *H* by the action of the genetic operators on other schemata in the population, and thus provides a pessimistic estimate of schema growth.

The related BBH of Goldberg [ibid.] maintains that GAs discover low-order schema of high fitness first, and then recombine these building blocks to gradually discover higher order, high-fitness schema.

Schema theory has been criticised since it says nothing about the reconstruction of schema by the positive action of genetic operators. The theory of GAs remains an active field. See Stephens and Waelbroeck [Stephens1999] or Langdon and Poli [Langdon2002] for an overview of work deriving exact predictive equations based upon the schema concept.

Evolutionary methods have been successfully applied to many problem domains, including control systems, data mining, game playing, machine learning, and scheduling. As we shall see in Chapter 3, they have also been successfully used to generate controllers for mobile robots. One advantage of evolutionary methods is that they are *population based*. The individuals in the population sample many points in the search space concurrently, and search is improved by swapping information between individuals by means of the genetic operators. This may be particularly useful when compared with non-population based techniques in deceptive problems, i.e. problems in which the search space is multi-model with many small peaks of fitness that are not the globally optimal fitness.

Another justification for using evolutionary techniques is that the core algorithm has proven its remarkable utility in the last 3.5 billion years of life on Earth. The behaviour of the living organisms that surround us and their multitude of intricate design fitting form to function is ultimately due to physically grounded and embodied evolution.

1.2.3 Evolutionary Algorithms for Multiple Objectives

Most real-world problems involve multiple objectives that may conflict. There has been a considerable amount of work on multi-objective optimization problems, and numerous researchers have reported success through the application of evolutionary techniques; see [Zitzler1999], [Coello2000], or [Fonseca1995] for an overview.

The majority of this work concentrates on generating a population of candidate solutions using an evolutionary algorithm, such that each of the candidate solutions is 'Pareto Optimal'. That is, they are members of the set of solutions for which there exists no solution that is better in respect to one objective, without being worse in respect to another, and which still obeys all solution constraints. The 'decision maker' then picks a solution from this set of *non-dominated* solutions.

A number of different evolutionary algorithms have been used, for example, various formulations of genetic algorithms, e.g. [Deb2000] and genetic programming, e.g. [Rodriguez-Vazquez1993].

Evolutionary Algorithms are especially suited to the problem of discovering members in the Pareto optimal set (i.e. the set of non-dominated solutions). Firstly, they are population-based, and thus can maintain information about many points on the Pareto front[5] simultaneously. Secondly, they are less susceptible to the shape of the Pareto front; concave or discontinuous Pareto fronts pose difficulty to more traditional mathematical approaches [Coello2000].

[5] The Pareto front is the part of the boundary of the set of all solutions which comprises the Pareto Set.

A number of approaches have been used to apply genetic algorithms to multi-objective optimisation, for example:

- Derive a fitness function that combines the multiple objectives into one function. The multiple objectives need to be weighted relative to each other. This may be difficult in problem domains about which little is known.
- VEGA (Vector evaluated GA) [Schaffer1985]. Each generation the population is assessed for performance against each of the objectives in isolation, and the total population split into k sub-populations where k is the number of objectives. Individuals from these sub-populations are then intermingled again, and bred. This approach suffers from the problems of ‘speciation’, whereby there may arise sub-populations which specialise in one objective.
- MOGA (Multi-objective GA) [Fonseca1993]. The *rank* (r_i) individual of the population is set to 1+n_i, where n_i is the number of individuals by which it is dominated. Nondominated individuals are assigned the rank 1, and individuals at the same rank have their fitness assigned and averaged in such a way that nondominated individuals have higher fitness. In order to maintain a population of diverse nondominated solutions a nicheing mechanism which considers the distance between any two individuals assigns higher fitness to individuals in less-crowded areas of the search space. Fixing this sharing parameter is the chief difficulty in applying MOGA to problems.
- NSGA (Nondominated Sorting GA) [Srinivas1994]. Similarly to MOGA, NSGA and NSGA-II [Deb] use a ranking scheme on the basis of level of domination to assign fitness, and also a nicheing strategy. Although NSGA is less computationally efficient that MOGA, the later NSGA-II has improved efficiency and maintains a better spread of nondominated solutions in the final population.

It will be seen that there are two goals in multiobjective GAs. Firstly, to discover candidate solutions in the Pareto-optimal set that are nondominated. Secondly, to maintain as diverse a population as possible within the set of nondominated solutions. The latter is important because the trade-offs that enable the experimenter to choose between solutions are not represented within the fitness assessment; all nondominated solutions are candidates which solve the problem.

It would be possible to use multiobjective evolutionary algorithms to produce robotic controllers. However, as shall be shown in Chapter 3, using GAs alone in a robot context is generally a slow process, since fitness evaluation on the physical platform is time consuming and must be repeatedly performed for all individuals in the population. Essentially, GAs are an *offline* learning method, since the controller produced by each individual must run in order to establish its fitness, and while this is taking place no knowledge is incorporated into the system. Also, the second goal of multiobjective GAs of maintaining a diverse population from which an experimenter may choose a representative is unnecessary in order to develop a single optimal controller, and introduces an additional human step which reduces the autonomy of the system.

For further information on the subject of evolutionary multiobjective optimisation the reader is referred to Deb's book for a comprehensive overview [Deb2001].

1.3 Reinforcement Learning (RL)

Natural organisms display adaptation in many ways. We have briefly mentioned the process of evolution by natural selection, thought to be responsible in some part for the diversity of form and behaviour observable in the natural world. Other forms of adaptability affect the phenotypes. For example, the ontogenetic processes of development by which the phenotype is created from the genotypic encoding adapts to changes in its environment, and such plasticity of growth can generate many different forms from the same genomic starting point. In addition to the adaptation of evolution which takes place over many generations, animals also adapt through learning within their lifetime. The study of learning provides another starting point for the simulation of adaptive behaviour.

Learning mechanisms can be categorised according to whether they are *supervised* or *unsupervised*. Supervised learning implies the concept of teaching by example; the learner is presented with examples for which the correct result is known, and after a training period can then predict results for new observations. In unsupervised learning an agent learns through interaction within its environment. Reinforcement learning is one such unsupervised learning methodology[6], which draws upon biological examples and is represented by a number of different classes of algorithm.

1.3.1 An Overview of Reinforcement Learning

Skinner, the behaviourist student of animal learning, built on the work of Thorndike and others to develop a theory of learning known as *operant conditioning.* In operant conditioning, an animal learns to associate a reward (an unconditioned stimulus (US) – i.e. something the animal innately 'desires' - such as food) with performing an action (an

[6] RL in single step problems is very similar to supervised learning, however, in RL the solutions are not directly manipulated to reduce observed error.

unconditioned response (UR)). A conditioned stimulus (CS) such as a light is also presented at the same time as the US. After a training period the animal will produce the UR in the absence of the US upon perceiving the CS. This is contrasted with the experiments of Pavlov in *classical conditioning* in which an US that **triggers** a companion UR (e.g. as the perception of food triggers a dog to salivate) is presented at the same time as a CS, and after training the animal will produce the UR upon the presentation of the CS alone.

In essence, operant conditioning is learning by *trial and error*. The animal learns to associate a combination of environmental cues and behaviour with a reward. Thereafter it is more likely to perform those actions that lead to higher reward in that particular environmental state.

This is the model followed in reinforcement learning. The environment is presented to the learner in some unambiguous way, so that the same environmental state can be recognised when it is encountered again. There is also some measure of reward from the environment. In response to the environmental information, the learner chooses an action according to its *policy*. After following the action advocated by the policy, the environmental state and reward are again supplied to the learner.

For the value of an environmental state to be determined, the environment must be rendered in some way into distinct states. The state signal cannot be expected to inform the learner of all useful information about the environment, but it should summarize past states compactly so that all relevant information necessary to make optimal decisions is presented to the learner. Such a state signal is said to have the *Markov Property*, or to be *Markov*. More formally, if the state has the Markov property, then the environment's response at time *t+1* depends only on the state and actions representations at time *t*. The environment as a whole is considered Markov if this is true for all environmental states.

Modern reinforcement learning implementations can be divided into three approaches. The earliest, dating from the work of Richard Bellman in the 1950s, is *dynamic programming (DP)*. Dynamic programming requires that a model of the environment is perfectly known in order to iteratively estimate the value of states. For this reason (and its great computational complexity) it is of little utility in the problems considered herein where an entity has no *a priori* knowledge of its environment. The second approach is known as *Monte Carlo methods* which do not require a perfect model but can learn from experience. Monte Carlo methods build knowledge in an episodic fashion – they require that a sequence of steps between Markov states has a defined termination, at which point the values of the states leading to that final state can be calculated. Since this may not be true in the case of a robot that has to continually perform tasks, and should optimise its behaviour along the way without the concept of episodes, Monte Carlo methods are not dealt with further in this thesis.

The third approach is a synthesis of Monte Carlo methods and DP, and is known as *Temporal Difference (TD)* learning. It requires no perfect model, and can update value estimates as new states are experienced, guessing the values of states from other guesses. The values estimated are either the values of states themselves $V(s)$, or the values of taking a particular action when in a particular state $Q(s,a)$.

TD methods can be divided into two categories. *On-policy* methods evaluate the values of a policy while using it for control. *Off-policy* methods use a different policy to choose their actions from the one which is being evaluated.

Sarsa [Rummery1995] is an on-policy TD control method which learns an action-value function rather than a state-value function. As it learns the value of state-action combinations, it can be used to control an agent's movements around an environment. Its update function is given below;

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

in which α is the *learning rate*, γ is the *discount rate* (both in the range [0,1]), and r is the *reward* from the environment. Sarsa gets its name from the tuple (s_t, a_t, r, s_{t+1}, a_{t+1}), since all elements are used in the update equation. It has been proven that Sarsa will converge to the optimal policy given that all state-action pairs are visited and their corresponding Q values are updated indefinitely.

Q-learning [Watkins1989] is an off-policy method. The learned action-value function Q directly approximates the optimal action-value function, Q^*, independent of the policy being followed.

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

In contrast to Sarsa which updates *Q(s,a)* for the policy it is actually using, Q-learning updates *Q(s,a)* for greedy policy with respect to the current *Q*. Sutton and Barto [Sutton1998] show an example in a simulated grid-world problem called 'Cliff Walking' in which Q-learning might achieve worse results than Sarsa due to its off-policy action selection. The reader is directed to Sutton and Barto's excellent book [Sutton1998] for further details on reinforcement learning.

In summary, reinforcement learning presents methods which allow for online learning of optimal behaviour. This would clearly be useful in problems of controlling robots. One problem with Sarsa and Q-learning is that they need to maintain Q-values for every state-action combination. If the state-action space is large, this may not be efficient. Furthermore, the necessity to render the environment into uniquely identifiable Markov states requires that these methods should either have an artificial *a priori* discretisation imposed upon the environment, or that some external generalisation mechanism is supplied which will perform additional unsupervised learning in order to collate real-valued sensor data into a smaller number of discrete states. We shall see some examples of robotic applications and the ways in which these problems have been addressed in Chapter Three.

There now follows a brief overview of some work in which reinforcement learning has been used in problems with multiple goals or objectives.

1.3.2 Reinforcement learning for Multiple Objectives

Mariano and Morales [Mariano1999] present work in which they used multiple co-operating families of Ant-Q reinforcement learners of the type first presented in [Gambardella1995]. Ant-Q is a distributed algorithm for combinatorial optimization using RL to solve the travelling salesman problem (TSP). Mariano and Morales extend this work to solve a TSP where there are a number of simultaneous objectives; in their application, this is an irrigation system which must be the least cost (i.e. the shortest path), and which places crops appropriately to the availability of water. They report better results using this method than in a comparable treatment where the irrigation system was optimized first, and crop placement second.

Crabbe makes the point that, in the case where an animat has multiple objectives to satisfy simultaneously, this cannot be optimally achieved through having separate Q learners for each objective [Crabbe2001], with the one with the highest activation determining the animat's action. He considers the example of a robot that has two objectives, which can be satisfied in any order; getting power, and getting a building block. If the environment is dynamic, there is a chance that conditions will change while seeking one goal such that the other goal can no longer be achieved - another robot might consume the battery, for example. Using Utility Theory, Crabbe shows that the best course of action is not only dependent on the expected value of the outcome, but is also dependent on the probability of success. Simply stated, in order to maximise the overall utility, sometimes one should pick the low-hanging fruit, although they may be less juicy!

In order to solve such problems, Crabbe [ibid.] states that one could either combine the output of a number of Q-learners, or have one Q-learner that can manage multiple simultaneous goals. He examines the latter case, and goes on to show that while Q-learning systems typically use a linear scalar reinforcement function, there can be circumstances in multi-objective problems where such a function cannot be used if the fitness landscape in respect for either of the objectives is not monotonic. However, a bi-linear function can be used with a single Q-learner since such a function is non-linear in respect to the combination between its linear variables.

Gabor et al. [Gabor1998] have considered similar multi-criteria sequential tasks. Examples of such tasks would include situations where a robot has to perform a number of tasks, but the way in which it performs task A can lower the value it will receive from performing task B. The example they give is of ‘Buridan's Ass’; placed between two dishes of food its hunger drives it to one or the other, but in doing so it raises the possibility that the food on the other dish will be stolen. Since the ass tries to optimize his overall utility, he guards both dishes, thereby eating nothing. He therefore has two different conflicting objectives.

Gabor et al. present a method of determining how to mix behaviours such that the total loss due to conflict is reduced, without reducing the problem to a scalar-valued reinforcement case. Using a vector-valued reinforcement signal is possible if policies are compared component-wise; but as noted above by Crabbe, this may in some cases give sub-optimal solutions. An alternative approach is to use the weighted sum of the components using their evaluation functions; this approach reduces to the scalar-valued reinforcement case if the discount factor is the same for each criterion. They present a method that uses the framework of abstract dynamic programming, with vector-valued reinforcements, which can be used when there is no natural weighting between the components, and demonstrate its abilities to learn against a number of different players in the game of tic-tac-toe, its criteria for optimality being both winning, and reducing the number of moves made.

Mannor and Shimkin [Mannor2004] also consider the case of vector-valued reinforcement in a dynamic environment, but they use a geometric approach to steer the learner through the space of possible policies until it is within a target region. This approach is based upon the theory of approachability for repeated matrix games with vector payoffs. Consider the task of a thermostat, which can be viewed as navigating through a one-dimensional space to stay within certain bounds. More criteria increase the dimensions of the space through which the controller moves, the target region being the bounded n-dimensional space where all variables are within acceptable limits. Note that this work deals with multiple criteria optimization, rather than multiple (conflicting) goals.

1.4 Learning Classifier Systems

From the production systems that stemmed from GPS came *classifier systems,* first described in the form of *CS-1* (Cognitive System One) by Holland and Reitman [Holland1978]. Classifier systems added the concept of schemas to the production rules, thereby allowing generality in matching. Within the system proposed by Holland and Reitman there were two forms of learning; a ‘simple learning process’ and a ‘…more complex learning algorithm’.

The latter was the GA which Holland had previously described [Holland1975]. The former is briefly described;

> *'When a reward enters the system …the predicted payoffs of the currently activated classifiers are then modified to reflect their* ***accuracy*** *in anticipating this reward. Those predicted payoffs that were consistent with (not greater than) this reward are maintained or increased; those that overpredicted are significantly reduced…'* [Holland1978, emphasis added]

CS-1 will not be described in any detail here, although we will return to it later. Instead, below is presented a brief (and greatly simplified) overview of Holland's '*Michigan style*' Learning Classifier System (LCS), as the systems which stemmed from CS-1 became known. Michigan style LCS [Holland1986] apply a GA to a population which is a set of rules, or 'classifiers'. The whole population functions essentially as one production system. This is in contrast to '*Pittsburgh style'* LCS [Smith1980] in which the population is a set of rule sets. Each individual, then, functions as a production system, and is evaluated by applying it for a short length of time to some problem and rating its performance.

Although Pittsburgh classifier systems are more firmly grounded in GA theory, Michigan classifier systems have an additional mechanism of credit assignment resembling online reinforcement learning. For this reason they are likely to learn quicker, since they do not require the performance of every individual in a population to have its fitness assessed before adaptation takes place.

1.4.1 An overview of Michigan-style Learning Classifier Systems

The Michigan-style classifier system operates on a population of rules or classifiers. These take the form of

- Condition; a string of length *k* drawn from the alphabet {0,1,#}[7].
- Action; a string of length *k* drawn from the alphabet {0,1,#}.
- Some measure of the classifier's 'strength'.

In the condition, a '#' symbol has the familiar meaning from GAs of 'don't care'. In the action, a '#' symbol means 'pass through'. We shall examine the meaning of this below.

The LCS incorporates two types of learning: a GA which searches the spaces of rules, generating novel rules and increasing the proportion of the population composed of higher fitness rules, and reinforcement learning which alters the fitness of rules according to their role in achieving reward. In Figure 1-1 the GA is shown as the 'Rule Discovery System', and the reinforcement learning system is shown as the 'Credit Apportioning System'. These components, with the third 'Performance System', are described below.

[7] Holland states that a classifier might also have an arbitrary number of conditions [Holland1986].

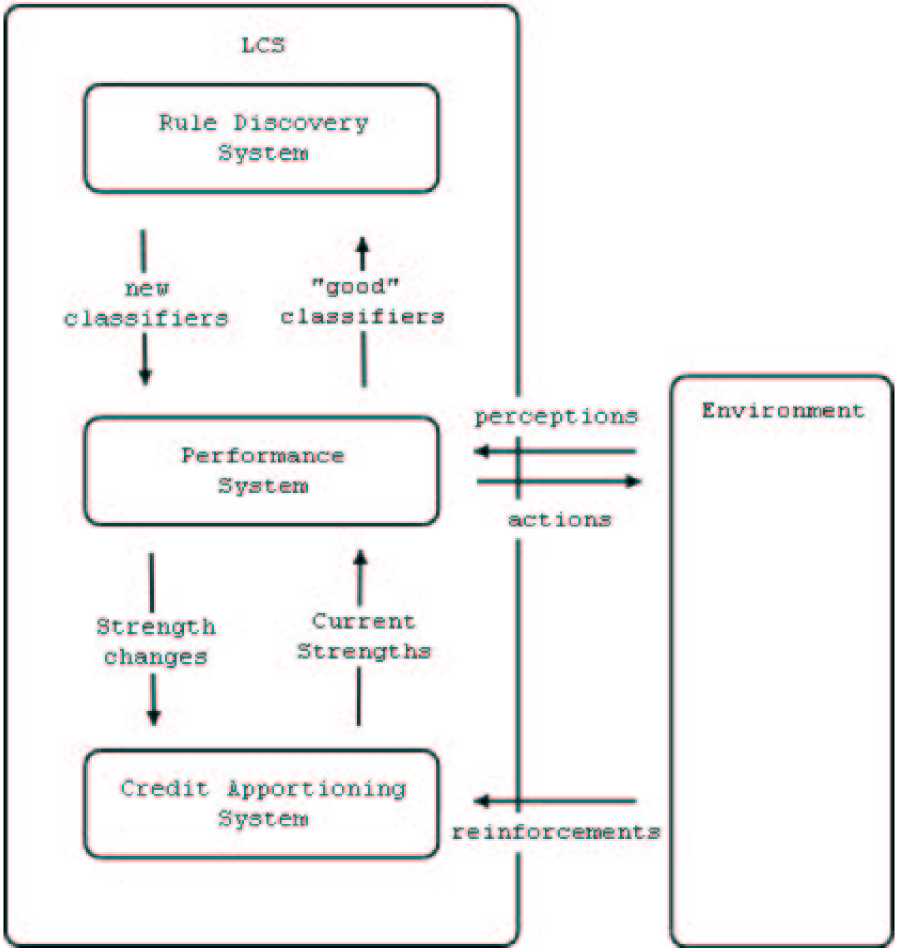


Figure 1-1 The main components of a LCS, from Dorigo and Colombetti [Dorigo1998]

Performance System.
The Performance System of the LCS consists of three parts.

- Interfaces. Environmental input is presented via detectors, and the actions advocated by the performance system is carried out via the effectors.
- Message List. Processed environmental representations of length *k* in the alphabet {0,1} are placed on the message list, where they can be matched by the condition part of the rules. When a condition matches any message on the message list, it places a message on the message list itself. This message will be the action of the rule. Where a '#' pass-through symbol occurs in the action, the corresponding character of the message matched by the condition is copied at that position.

- Rule Base. The population of rules, or classifiers.

The basic execution cycle is thus:

1. Place all messages from the input interface on the current message list.
2. Compare all messages to all conditions and record all matches.
3. For each match generate a message for the new message list.
4. Replace the current message list with the new message list.
5. Process the new message list through the output interface to produce system output.
6. Return to step 1.

Because the condition of one classifier can be triggered by the message posted by another classifier, ‘semantic networks’ of classifiers can be implemented. More general classifiers, i.e. those with a greater proportion of don’t care symbols, are likely to be part of more such networks than more specific classifiers.

The classifier system thus far described does not change in the composition of its rule base during operation. In a *learning* classifier system, two further systems are introduced

Credit Apportioning System

The credit apportioning system operates via the 'Bucket Brigade' algorithm, inspired by a simplified model of economics. This changes the activities in the basic execution cycle outlined above. In order to get its messages onto the message list when its condition matches any of the current message list, the classifiers bid in an auction. A classifier's bid is proportional to its strength, and also to its specificity. The winning classifier pays part of its strength to the classifiers that posted the messages which its condition matched; the amount received by such classifiers is the amount of the bid divided (equally in the simplest case) between them. At each iteration, all classifiers may also pay a tax, and have their strength reduced slightly.

In this way each classifier can be seen as '*…a kind of middleman in a complex economy…*' dealing only '*… with its suppliers – the classifiers that send messages satisfying its condition – and its consumers – the classifiers that are in turn satisfied by the messages it sends'* [Holland1986]. The ultimate consumers are those classifiers that post messages leading to external actions via the effectors which directly lead to reinforcement signals being received from the environment. The bucket brigade transfers the reinforcement signal back, step by step, through the supply chain of classifiers which participated in the eventual action which led to external reward.

This then provides for online reinforcement learning by the system. However, the ability of the system to solve problems through the implicit chaining of classifiers in the way outlined so far depends on the identities of the classifiers in the initial rule base. In the case where the initial rule base does not encompass suitable classifiers, the system cannot function optimally. Holland therefore introduced the third component, the Rule Discovery System.

The Rule Discovery System

The Rule Discovery System is similar to a standard GA, as described in Section 1.2.1. The population of classifiers in the rule base have a chance to reproduce proportional to their strength, which therefore also serves as a measure of their fitness. In order to preserve knowledge, the population is not changed generation by generation, but instead a steady-state GA is used. The offspring of parents chosen by fitness-proportionate selection replace the weakest members of the population, as the latter are likely to be classifiers which have participated little in chains of rules leading to external reward.

1.4.2 Why use Learning Classifier Systems for Robotics?

It will be clear from the description of Holland's LCS given above that classifier systems can be very complicated. Holland's original LCS, and implementations based upon it, proved to be subject to a number of weaknesses, and difficult to implement. The chaining of rules together in the bucket brigade was prone to exploitation by very general rules; a rule that is part of many chains may receive more reward. Dorigo and Colombetti found that their implementation of the system suffered from '.. *rule strength oscillation, difficulty in regulating the interplay between the reinforcement system and the background genetic algorithm (GA), rule chains instability, and slow convergence*' [Dorigo1998].

Fortunately, there exist today a number of greatly simplified LCS. Stewart Wilson first introduced such a version of Holland's system, which he named ZCS [Wilson1994]. This was closely followed by XCS, a classifier system based on accuracy [Wilson1995]. In both these systems the link to the methods created within the reinforcement learning community was more clearly recognised and implemented. The work in this thesis applies both ZCS and XCS to problems of learning optimal multiobjective robotic control, and detailed descriptions of the operation of both algorithms will be given in later chapters. It is the author's belief that there are a number of reasons for choosing a modern Michigan-style LCS for this task, rather than a purely evolutionary method, artificial neural networks, or a pure reinforcement learning method.

- Unlike a purely evolutionary approach, the addition of reinforcement learning allows for online learning. Learning is incorporated into the system while the task is being performed, not episodically after the assessment of all individuals' fitnesses.
- Unlike an artificial neural network in which the weights of the connections between the neurons encode the learning of the system in a fashion which is difficult to understand, the rules of the classifier system offer the benefit of interpretability.
- Unlike the implementations of reinforcement learning for robotics, in which the necessity to render continuous space into a discretised Markov space involves either *a priori* decisions about the problem which may be wrong, or the addition of function approximation mechanisms such as neural networks, an LCS has the ability to generalise built in.

These points are expanded upon and further explained in Chapter Three.

There now follows a brief description of some work in which the concept of multiple objectives has been tackled within the LCS community.

1.4.3 Learning Classifier Systems and Multiple Objectives

The CS-1 algorithm described by Holland and Reitman [Holland1978] includes '*resource reservoirs*' which '*… reflect simple biological needs…*' and '*… deplete regularly in time*'. The authors note that these internal states could easily be used to model cognitive goals. Note that the resource reservoirs are not presented to the condition part of the classifiers as the 'internal environment', but instead influence the choice of which classifier's action is taken by the learner. A classifier's chance of being chosen is proportional to the specificity of its match to the environment, both internal and external, '*multiplied by the amount of the current needs fulfilled by this classifier's predicted payoff*'. The authors outline a simple linear environment where the system starts in the middle, and can receive at one end a reward that partially fills one internal resource reservoir, and at the other end a reward satisfying the other of these two internal needs. One reward is twice the size of the other, and the system is shown to improve in performance to visit the rewards in the correct two-to-one proportion while also minimising the number of steps taken to achieve this behaviour. They further show how the system uses past learning to cope with changing environments.

Dorigo and Colombetti [Dorigo1998] used a novel LCS, Alecsys, to control a robot. The main thrust of their work was to evolve simple behaviours, and then combine these to produce more complicated behaviours through a hierarchical system of LCS. A hierarchy is imposed on a problem and an 'LCS coordinator' is introduced to select an action from the LCS lower in the hierarchy. The lower level LCSs are trained or shaped on their task in isolation, and once an acceptable level of performance is obtained for each lower-level module the rules that it contains are fixed and no more learning takes place. When this happens the training of the higher coordinating module is performed. Results were obtained for various following/avoiding tasks in simple environments in which the coordinator switches between different behavioural modules as appropriate. This work will be examined more closely in Chapter Three.

Llora and Goldberg [Llora2003] present an approach to *Pittsburgh-style* classifier systems which is in itself multi-objective, since they consider at the same time the objectives of accuracy of the classifiers, size of classifiers, and the generality of the solution. In contrast with the work presented in this paper, their multiple objectives describe a balancing of desirable characteristics of the learning system, and then maintaining the Pareto front of solutions which balance these characters, rather than learning systems solving problems that have multiple objectives. It is essentially the same as the multi-objective optimization work mentioned above.

Chapter 2 ZCS

2.1 Introduction

The Learning Classifier System, introduced by Holland [Holland1976, Holland1986], was complicated and difficult to realize. Therefore, Wilson [Wilson1994] introduced the 'Zeroth Level Classifier System', ZCS. In this he made a number of simplifications to make the system easier to understand and to implement, '*while retaining what we deemed to be the essence of the classifier system idea*' [ibid.]. The most notable of these simplifications was the absence of an internal message list; therefore the system has no temporary memory. In this way, the system cannot act upon information stored in response to previous interactions with its external environment, nor can it internally generate 'drives' or 'intentions'.

This chapter investigates ZCS, showing that although it was initially believed to be incapable of optimal behaviour it has been shown to be capable of optimal performance in some environments, and review some related work in which ZCS's performance has been investigated. An examination of the use of ZCS in multi-objective problems of increasing complexity is presented thereafter.

For ZCS to perform optimally, it must have its parameters carefully set [Bull2002a]. We shall see one method by which such optimal parameter settings can be discovered, and explore the space of ZCS's performance in relation to the most important of these parameters.

2.2 ZCS Algorithmic Description

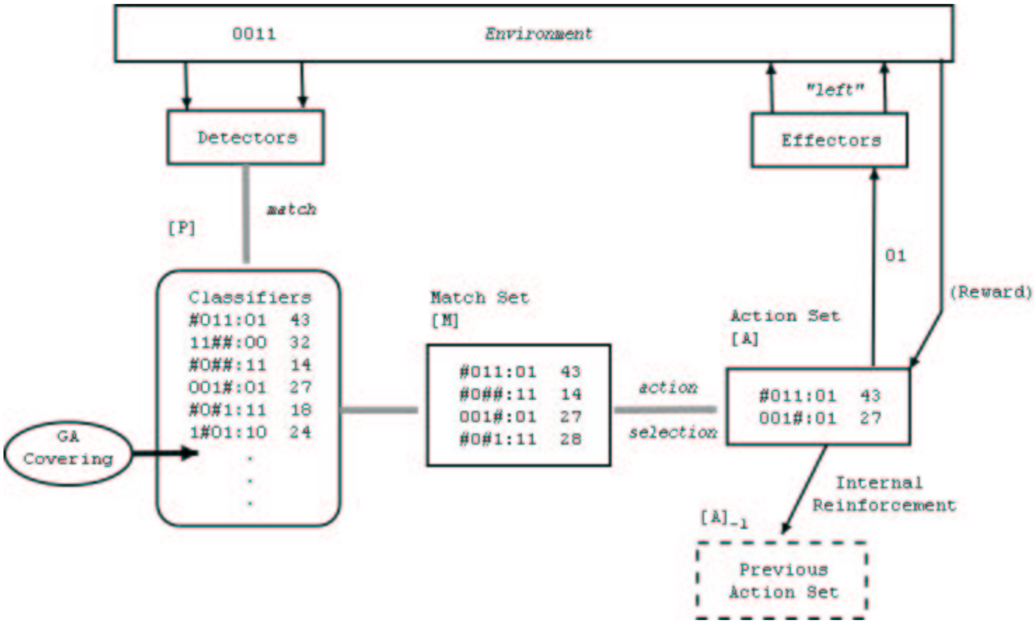


Figure 2-1 Schematic Illustration of ZCS after [Wilson1994]

As described in [Wilson1994], ZCS is a Michigan-style Learning Classifier System, without internal memory (Figure 2-1). It is presented with an encoded representation of its environment, and may also be presented with a scalar reinforcement signal, or 'reward'. The reward is contingent upon the system's success.

ZCS contains a population [P] of *N* classifiers. In Wilson's original formulation, classifiers have a condition and action part encoded using the ternary and binary alphabets as detailed in Chapter 1, with the wildcard '#' allowing generalization. Upon presentation of the binary encoding of the environment, the subset of [P] which matches is termed the 'match set' [M].

It may happen that no classifier in [P] matches the environmental input, and [M] is empty. In this case, a process known as *covering* takes place. A classifier is created *de novo*. Its condition matches the current environmental input, each character having a probability $P_{\#}$ of being replaced by a wildcard character. An action is chosen at random from the set of possible actions. The new classifier is assigned the average strength of the population, and replaces a classifier chosen from the population. This choice takes place through roulette selection on the inverse of the strengths, and therefore it is more likely that a weak classifier will be replaced in this way. Cover may also take place if the total strength of [M] is less than a certain pre-specified fraction ϕ of the mean strength of [P].

The classifiers which form [M] may be considered as hypotheses on which action to take in the current environmental state. One action must be chosen from the set of actions thus advocated. Although a number of selection schemes could be used, roulette selection on the strengths of the classifiers in [M] is used, such that classifiers of higher strength are more likely to be selected. In contrast to Holland's work, no measure of classifier specificity is involved. All the classifiers in [M] which advocate the same action form the *action set* [A]. The chosen action is then taken.

Reinforcement in ZCS occurs through redistributing reward through the action sets that lead to this reward. A percentage β of the strength of the classifiers in [A] is subtracted and placed in an initially empty 'bucket'. After the action has been taken, a fraction β of any external reward received is divided amongst the classifiers in [A]. A record is kept of the previous action set $[A]_{-1}$, and if this is not empty, the 'bucket' is discounted by λ and the then divided equally between its members, . In this way a discounted reward flows back through the 'implicit bucket brigade'. Additionally, the classifiers which were in [M] but not in [A] have their strengths reduced by a fixed fraction τ, the tax rate. This causes classifiers which advocated a different action from the one taken to be penalised. The interaction between the flow of reinforcement signals from the environment and tax rate will affect the balance of exploitation and exploration in the learning system, though it should be noted that the likelihood of generation of new classifiers by cover and through genetic ‘churn’ will also interact with this balance.

The ‘implicit bucket brigade’ algorithm outlined above was noted by Wilson to be similar in some respects to Q-learning [Watkins1989]. However, it is more similar to Sarsa, a TD method for learning action-value functions rather than state-value functions, as noted in [Sutton1998].

In addition to the adjustment of strengths through the process outlined above, there is a panmictic[8] genetic algorithm that replaces weak classifiers with copies of stronger classifiers. Parents and replaceable classifiers are once again chosen using roulette wheel selection. The new classifiers generated to replace the weak may undergo single point crossover with the probability χ and may mutate at any position in the condition or action part with the probability of μ per allele. The GA is invoked after all reinforcement has taken place with the fixed probability ρ. New classifiers generated by the GA are initialised at half their parent’s strength, and the parent has its strength reduced by a half. Therefore reproduction (in the absence of the genetic operators) does not in itself change the likelihood that an action will be chosen.

In Wilson’s original presentation of ZCS, he presented graphs showing that ZCS failed to achieve optimal performance on the Woods1 maze problem. He suggested that this was due to failure in balancing exploration and exploitation; in short, the algorithm has found a sub-optimal solution due to an incomplete exploration of the solution space, and fixated prematurely upon a solution. He suggested a number of possible enhancements to this classifier system, and thereafter presented his influential XCS accuracy-based classifier system in which the simplicity of ZCS is lost.

[8] Panmictic. Referring to populations in which there are no constraints placed upon breeding.

2.3 Related ZCS Work

Since ZCS provides a simpler platform on which to experiment with classifier systems, it has been used as the basis for work reported in the literature in a number of areas. ZCS has been used to simulate multi-agent co-evolution [Bull1998, Bull1999]. Recently, Bull [Bull2002b] has used ZCS as a platform on which to investigate lookahead and latent learning, drawing inspiration from the sources which have also driven the development of ACS, the Anticipatory Classifier System [Stolzmann1998]. ‘Lookahead’ learning refers to systems in which the reinforcement is dependent on the ability to correctly predict the environmental state resulting from taking an action, while ‘latent learning’ refers to the learning that occurs in the absence of reinforcement and that is not apparent until a reinforcement is introduced. Tomlinson and Bull [Tomlinson1998] extended ZCS to produce a ‘Corporate Classifier System’, in which a macro-level evolutionary operator allowed the formation of ‘corporations’ of rules. The resulting corporations – linked classifiers – can take control of the system for a number of time steps. This was compared against ZCS with a simple implementation of internal state, as described by Cliff and Ross [Cliff1994].

In Wilson’s original paper on ZCS, he mentions the potential to extend the expressiveness of the classifiers’ encoding. Using a simple ternary alphabet in a string, one can only express logical ‘AND’ relationships between the values of the encoded environment. Wilson mentions that Koza’s Genetic Programming [Koza1991] evolves Lisp S-expressions, and suggests that such representations might be used for the condition or action of a classifier. A number of alternative classifier representations have been explored, including Fuzzy Logic e.g. [Valenzuela-Rendon1991] and Artificial Neural Networks [Bull2002c].

2.3.1 Self adaptation of parameters

ZCS has 10 parameters, the values of which control its performance. Some of these parameter values are critical to the success of the algorithm, while ZCS is more robust with respect to others. It would be advantageous if the algorithm could itself discover the settings which are optimal for the problem at hand. One approach that has been used to automatically set parameters in Evolutionary Programming [Fogel1992] and Evolutionary Strategies [Rechenberg1973] is to evolve the mutation rate itself.

A similar method was used by Bull and Hurst [Bull2000b] to set the mutation rate for ZCS. Each classifier has an associated mutation rate, stored as a real number, and which is inherited by its offspring. The mutation rate is itself mutated using a Gaussian distribution, and the resulting mutation rate is applied to the classifier's condition and action as normal. They showed that mutation rates decreased as the population converges upon a stable solution, and report that the mutation rates of classifiers close to the goal decreased faster and were fixed sooner than those of classifiers further down the chain of reinforcement.

In [Hurst2001] they investigated self-adaptation of the parameters associated with the reinforcement process itself; learning rate β, discount rate λ and tax rate τ. Trying to use the same mechanism as for the mutation rate showed unpredictable behaviour, since the classifiers adapt their own parameters selfishly, for example by reducing their value of β to place less of their strength in the common bucket. To avoid this, the learning rate at time *t* is determined by roulette wheel selection upon strength from the learning rates of the classifiers in the action set at t_{-1}. The reverse is true of the discount rate; it is obtained in the same way from [A] and applied to [A$_{-1}$]. They showed that self-adaptation of the reinforcement parameters can be used successfully. This is of obvious significance since the optimal settings of parameter values is not generally known *a priori*, and require considerable trial and error on the part of the experimenter. However, the authors do not report optimal performance.

2.4 Parameter Sensitivity and Optimal Performance.

ZCS is capable of optimal performance in the Woods1 grid-world in which ZCS was reported as sub-optimal [Wilson1994]. Bull and Hurst showed that ZCS can achieve optimal performance using a model of its performance based on simple difference equations, and demonstrated this empirically [Bull2002a]. The crucial difference was in the parameters which they used, see table 2.1 for a comparison with the parameters used in [Wilson1994].

Table 2-1 : ZCS Parameters for optimal Woods1 performance

	Wilson1994	Bull2002a
N	400	-
$P_{\#}$	0.33	-
S_0	20	-
β	0.2	0.8
γ	0.71	0.02
τ	0.1	-
χ	0.5	-
μ	0.002	-
ρ	0.25	-
ϕ	0.5	-

It is important to note that the 'online' performance of ZCS using these parameters still falls short of optimality. In order to make a clear comparison with XCS, in which of the alternate 'explore' and 'exploit' trials only the performance on deterministic 'exploit' trials is shown, Bull and Hurst ran ZCS in a 'deterministic mode' at the end of training. During this period, reinforcement continues, but the GA is switched off, and actions are chosen based deterministically on their strengths rather than by the use of the roulette wheel.

Bull and Hurst note that it is the fitness sharing in ZCS which enables the algorithm to avoid over-general classifiers from dominating the population. Consider the case of a classifier which is the action set at the goal, and receives a high payoff, but which also matches an environmental input far from the goal where general payoff levels after successive discounts are low. It might be expected to dominate its fellows in this latter niche in a simple strength-based system like ZCS. However, when a classifier is in [A] it has its strength reduced by β times its strength, and all classifiers in [A_{-1}] are rewarded equally from the discounted bucket. With time, as noted in [Wilson1987] all rule fitnesses tend towards the same value.

ZCS has many parameters. Unfortunately, there is a high degree of linkage between these parameters, such that it is not necessarily possible to optimise one parameter in isolation from the others. This makes it difficult to find a parameter set with which ZCS can solve a particular problem. Bull and Hurst [Bull2002a] further note that, since the implicit bucket brigade is a form of TD learning, there is a time delay in the operation of the fitness sharing process, and 'it is well known that sub-optimal solutions can arise **if the learning rate and/or discount rate are incorrect** for a given task' (emphasis added).

2.5 Single Objective Problem : Woods1

The Woods1 environment shown in Fig. 2.2 is a 'grid world' consisting of a toroidal array of 5 x 5 cells. Each cell may be empty space into which the animat can move, and in one of which it is placed at the start of each individual trial. A cell may also be filled by a 'rock', which prohibits a move onto that cell, or may instead contain 'food'. When the animat moves into the 'food' cell, it receives an external reward, and a new trial starts.

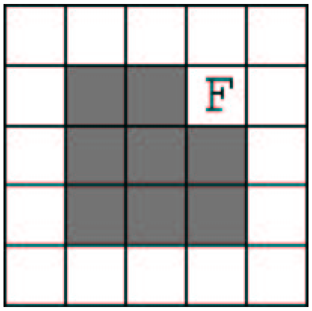


Figure 2.2-2 The Woods1 grid-world

Each environmental state is encoded by a unique combination of two characters. The state of the eight cells - starting at North and moving clockwise - surrounding the animat are represented as the 16 position character string resulting from the concatenation of these two character representations. In order to match this, the classifiers have a condition of length 16. The animat can move in one of eight directions. Each of these is similarly encoded as eight unique strings of three characters, and thus the action part of the classifier has three characters since a classifier can advocate only one action.

As mentioned above, ZCS is capable of achieving the average optimum steps to goal in this environment, given suitable values of learning rate β and discount rate λ (Fig 2.4). All graphs are the average of 10 runs. There is a final deterministic phase lasting 20% of the number of ‘on-line’ trials. The optimum average steps to goal is about 1.7 steps.

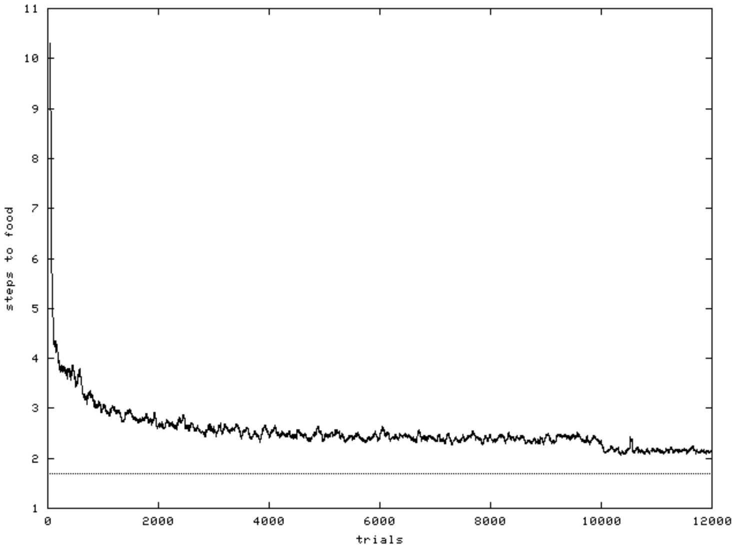


Figure 2-3 Woods1 with parameters from [Wilson1994]

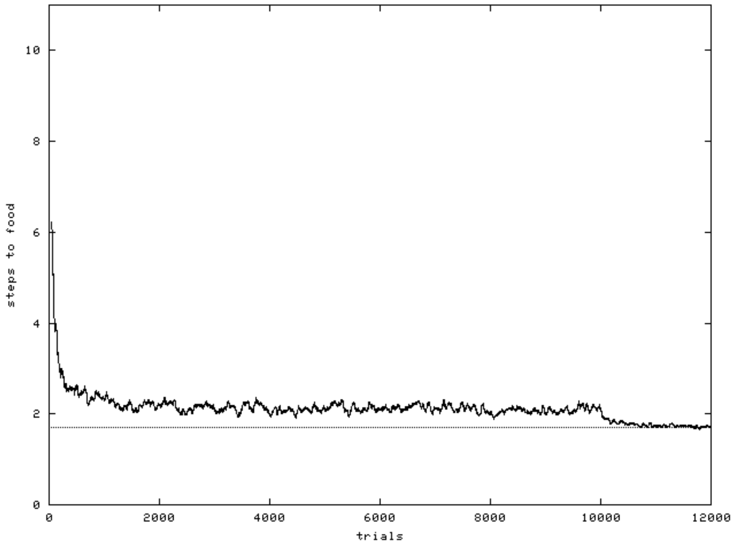


Figure 2-4 Woods1 with parameters from [Bull2002a]

Figure 2-3 and Figure 2-4 show the performance of ZCS in the Woods1 environment with Wilson's original parameters and those of Bull and Hurst. The latter enable optimality to be achieved, as mentioned above in Section 2.4.

2.6 Sequential Multi-objective problems.

Given that ZCS is capable of optimally solving the simple Woods1 environment with one objective, let us now examine multi-objective problems in this environment. Each is of increasing complexity. As mentioned in Chapter One, several sequential multi-objective have been examined by the reinforcement learning community.

2.6.1 Woods1 'Key and Door'

In the first such extension, the animat must visit a state in the environment before it reaches the goal. If it does not first visit this state, the trial will not terminate. This can be thought of as getting the 'key' to open the 'door' to reward. In order to allow ZCS to 'remember' whether the animat has previously visited the 'key' state, an extra character is added to the representation of the environment, this character being set from its initial state of zero to one when the animat visits the 'key'. This can be matched by a concomitant extra character in a classifier's condition. In Fig. 2.5, the 'key' is shown as 'K', and the reward state remains indicated as 'F'.

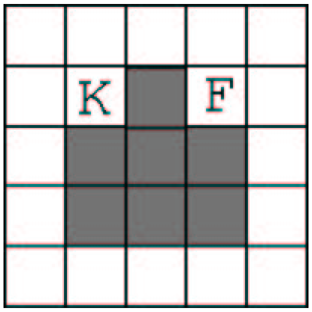


Figure 2-5 Woods1 'Key and Door'

Again, with suitably chosen parameters, ZCS is capable of solving this problem, as shown in Figure 2.6. The average optimum number of steps from any position in the environment to the goal state, going by way of the 'key', is approximately 3.7.

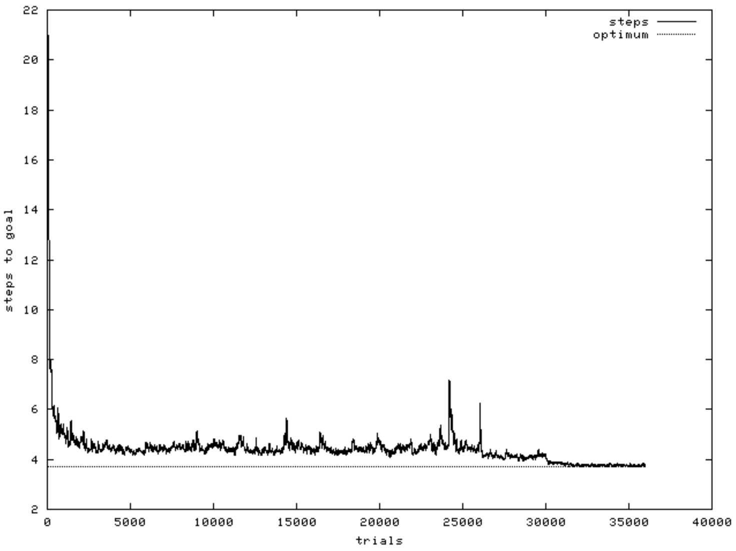


Figure 2-6 ZCS optimal for woods1 'Key and Door'

Here, the parameters chosen to run ZCS are as follows:

N=800, $P_{\#}$=0.33, S_0=20, β=0.97, γ=0.17, τ=0.2, χ=0.5, μ=0.002, ρ=0.25, ϕ=0.5

2.6.2 Woods1 ‘Carry the flag’

Of course, the above is a rather trivial problem. The animat can only achieve its reward and terminate the trial if it has first visited the key state. A slightly more complicated variant is provided by the same environment, in which the ‘door’ is always open. However, the reward gained is dependent on whether the animat has previously visited the ‘key’ state, being 1000 if it has, and otherwise 1. Once again, an extra character in both environment and classifier condition allows ZCS to ‘remember’ whether the animat has previously visited the ‘key’ state. Note that this ‘state character’ is **not** set by the classifier itself.

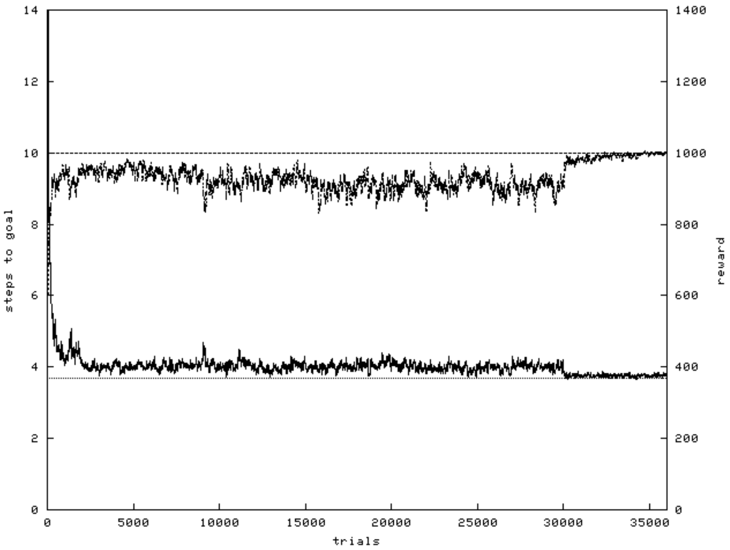


Figure 2-7 ZCS optimal for Woods1 'Carry the Flag'

Figure 2.7 shows that ZCS can successfully solve this slightly more complicated sequential multi-objective problem. All parameters remain as for the ‘key and door’ task, except: β=0.985, γ=0.2. As can be seen, steps to goal falls to the optimum average, and the average reward gained (windowed over 50 trials) rises rapidly, eventually reaching the optimal 1000 in the deterministic phase.

2.7 *Concurrent Multi-objective problems. Woods1e*

2.7.1 Fixed Cost, Stepwise Reward (woods1e-type1)

It is more interesting to consider the problems faced by a learner which has to juggle multiple simultaneous objectives. In order to do this, an alteration of the previous Woods1 environment is presented which not only has a 'food' goal, but also has another goal which is labelled 'energy'. In a similar fashion to the second sequential multi-objective task, the environment is presented to the classifiers with an additional character which is set to one if the animat's 'energy level' is higher than 0.5, and otherwise set to zero. The classifiers once again have an additional character in the condition which allow them to match this extension of the environment.

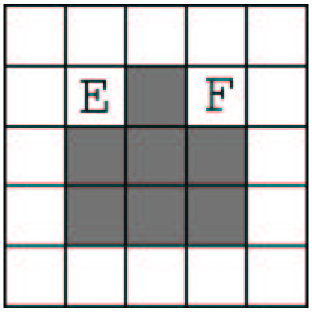


Figure 2-8 The Woods1e environment with two goals

At the start of the trial, the animat's energy level is set randomly in the range [0,1], so that alternate trials start with energy levels of less than, or more than, 0.5. The reward function is step-wise, giving an external reward of 1000 in the case that the animat arrives at the energy goal when its energy level is lower than 0.5, and a reward of 1 if the internal energy level is higher than 0.5. Conversely, the animat receives a reward of 1000 if it arrives at the food goal when its energy level is higher than 0.5, and is given a reward of 1 if its energy level is lower than 0.5. This problem and the Woods1e environment were first reported by Bull and Studley [Bull2002c].

There is no cost associated with movement. ZCS proves capable of optimally solving the Woods1e environment with no cost of movement and a step-wise reward function.

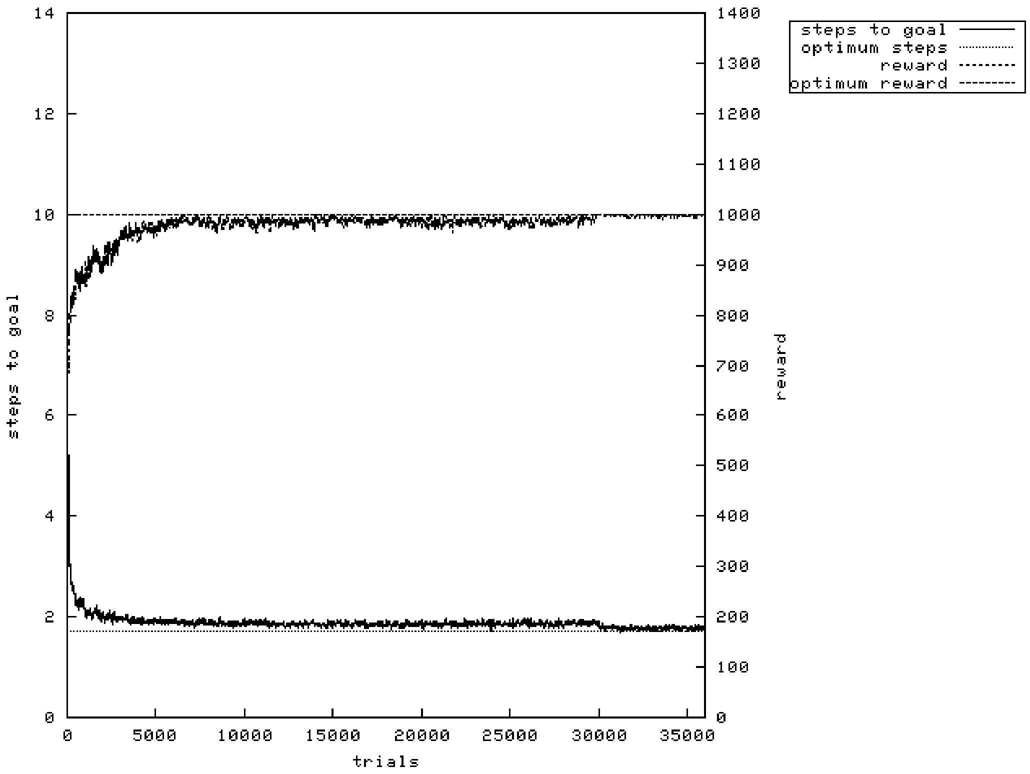


Figure 2-9 ZCS achieves optimal steps and reward in Woods1e with stepwise reward function and no cost of movement

As can be seen from Figure 2.9, steps to goal falls swiftly towards the optimum of 1.7, and the average reward gained rises to approach the optimum of 1000. In the deterministic phase, the off-line behaviour of ZCS shows that the optima have indeed been reached. All parameters remain as for the 'key and door' task, except: $\beta = 0.85$, $\gamma = 0.05$.

2.8 Achieving Optimality

As Bull and Hurst discussed, the most important parameters in determining the optimality of ZCS's performance are the learning rate β and discount rate λ. The other parameters are of varying importance; clearly, too high a rate of genetic churn due to χ, μ and ρ might be deleterious to the stability of the system, while too low a rate might make the discovery of solutions less likely due to incomplete exploration. However, it would appear from the experiments described in this thesis that the other main parameter upon which optimality depends is population size *N*. Cliff and Ross discuss the role of *N* in relation to their work on ZCSM; ZCS having internal registers that may be set through an additional 'internal' action part of the classifiers, and which are matched by an additional 'internal' condition. As the number of possible combinations of state and action increases, so a larger number of classifiers is needed in order to represent all combinations of state and action. Since ZCS seeks only the highest reward combination of states and actions, it is not necessary for all state-action combinations to be represented; instinctively however, it would seem that for a given problem there must be a happy medium that allows exploration though genetic churn and cover, yet simultaneously allows the maintenance of good candidate solutions in the population. In summary, in the author's experience, the main parameters which must be adjusted from Wilson's original settings are primarily β and λ, then *N*, and less importantly χ, μ and ρ.

Given that, for a sufficient value of *N* one can hope to find optimal performance through the manipulation of β and λ, it must be hoped that finding the 'sweet spots' in the space of ZCS performance with respect to these parameters are not akin to finding a needle in the proverbial haystack. If the performance landscape is rugged there is little hope of hitting upon optimal parameter settings.

2.8.1 Exploring the Performance Landscape.

Given that ZCS can solve the single objective Woods1 task with a population of $N = 400$, a population of 800 was used in the dual objective problems described above. This is because the problem may be seen in crude terms as two over-lying Woods1 trials, where the goal is in either one position or the other (see Figure 2-10 and Figure 2-11). Since the extra environmental character representing the value of the internal energy level compared with the threshold value of 0.5 effectively determines which goal is optimal at any time, it may be impossible for a classifier to generalize with respect to this character in some environmental states and still lead to the optimally parsimonious solution. In fact, examination of the environment shows that only three states[9] can be optimally generalized in this way.

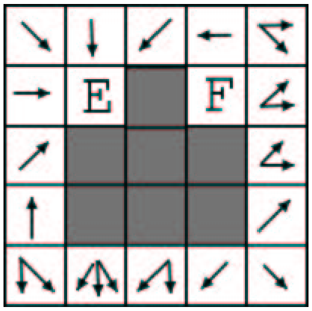


Figure 2-10 Optimal moves in Woods1e 'energy trial'

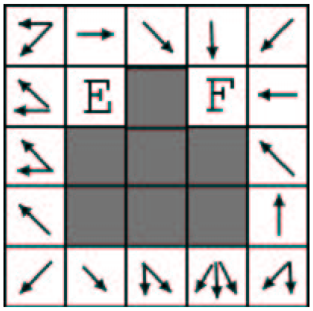


Figure 2-11 Optimal moves in Woods1e 'food trial'

Given the linkage between the parameters β and λ, it is difficult to hit upon the combination which will optimally solve a given problem, since one cannot adjust either in isolation.

One approach then is to explore the whole space of these parameters, iterating through the various combinations of β and λ with large increments between successive settings of each. This gives a coarse-grained view of how the performance of ZCS varies with respect to β

[9] Numbering from the top left = 0, 0, the cell at 2,4 can be generalized to 'South', 1,4 to 'Southeast', and 3,4 to 'Southwest'

(learning rate) and λ (discount rate), all other parameters being constant, and these views are intended to illustrate general trends. Figures 2.12 and 2.13 show respectively how the number of steps to goal, and reward gained, are influenced by these combinations of parameters. All points are the average of ten runs. Coloured contour lines on the x, y plane are b-spline interpolations of isometric values on the surface. All experiments have: N=800, $P_{\#}$=0.33, S_0=20, τ=0.2, χ=0.5, μ=0.002, ρ=0.25, ϕ=0.5.

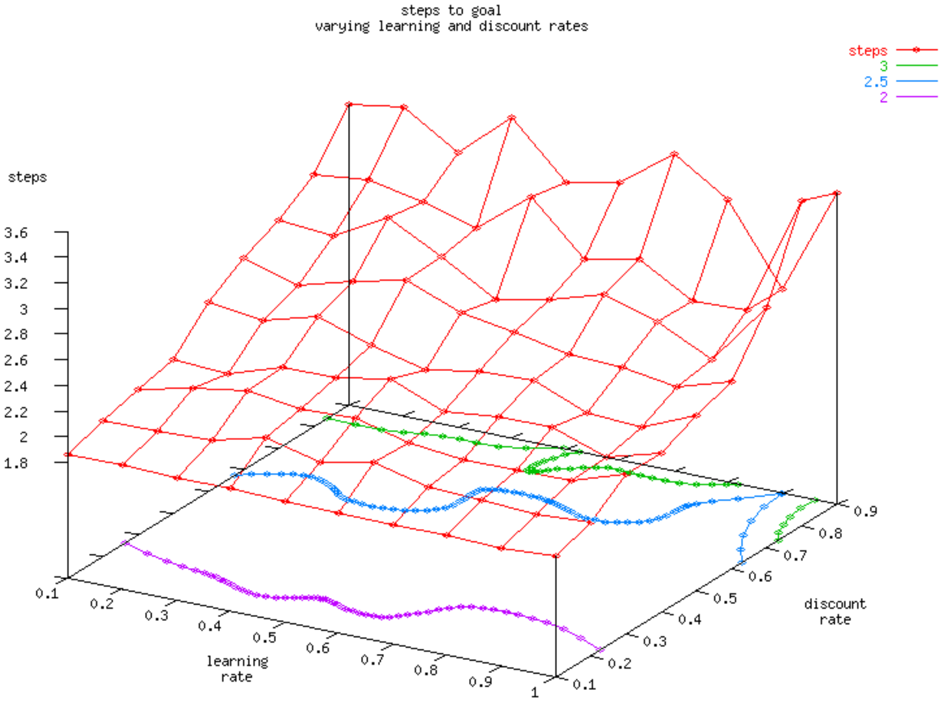


Figure 2-12 Coarse-grained exploration of steps to goal w.r.t. learning-rate and discount-rate in Woods1e with stepwise reward and no movement cost.

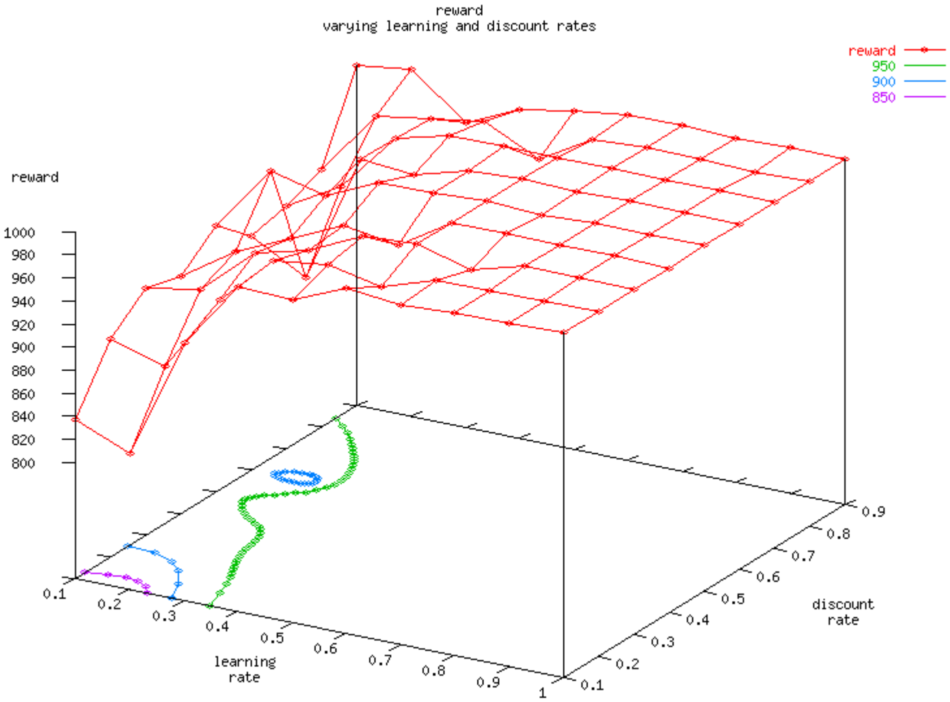


Figure 2-13 Coarse-grained exploration of reward gained w.r.t. learning-rate and discount-rate in Woods1e with stepwise reward and no movement cost.

These graphs reveal that, for all values of β, the average steps to goal in the deterministic phase reduces towards the optimum with lower values of λ. Lower values of λ decrease the amount of reward that flows back via the implicit bucket brigade to niches further away from the goal states. This may encourage more parsimonious solutions.

In contrast to the simple gradient we see with respect to steps to goal, the landscape with respect to average reward achieved in the deterministic phase is more complex. Here we see there is a plateau of optimality, irrespective of λ, for high values of β. As β drops, the landscape becomes more rugged; there is a higher degree of epistasis between β and λ with respect to reward than to steps to goal, allowing some points to achieve optimality while their near neighbours are distinctly sub-optimal.

In order to see where good candidate combinations of β and λ might be found, we may examine some measure of the overall performance of ZCS, taking into account both steps to goal (S) and the reward achieved (R).

$$\text{Performance} = ((\, R_{av} / R_{opt}) + ((\, S_{opt} - |\, S_{opt} - S_{av} \,|)/ S_{opt})) / 2$$

Where R_{av} is the experimental average reward, R_{opt} is the theoretical optimum reward, S_{av} is the experimental average steps to goal, and S_{opt} is the theoretical optimum steps to goal. Graphs of performance are presented with the axes of discount rate and learning rate swapped for clarity.

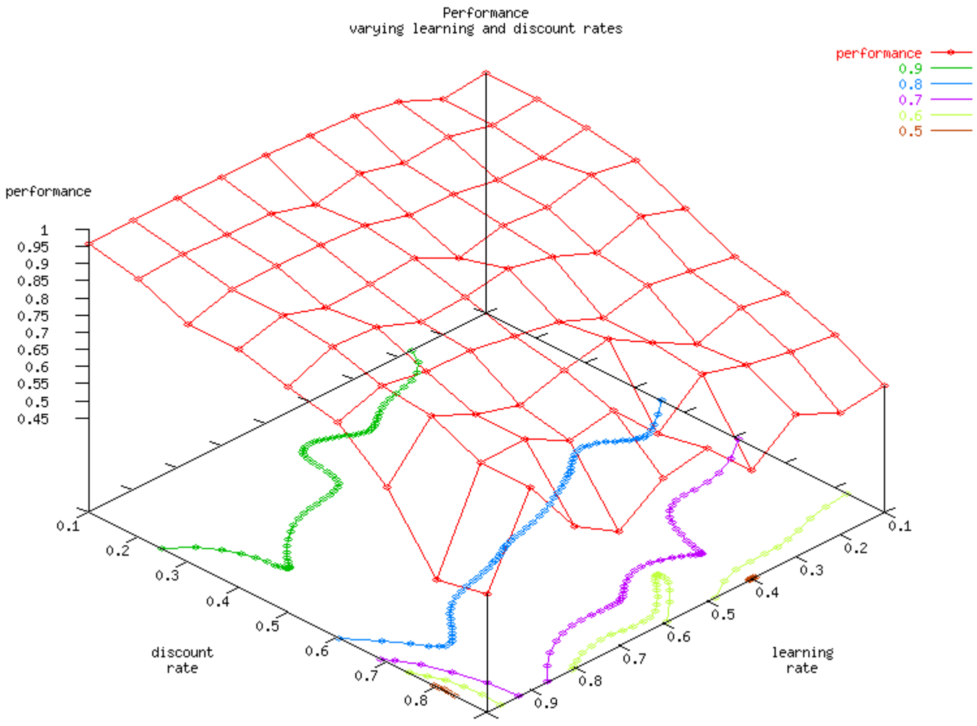


Figure 2-14 Performance of ZCS in Woods1 with stepwise reward and no cost of movement, for varying learning rate and discount rate

Noting the change in orientation of the axes, Figure 2.14 provides a clearer picture. We see that, for this problem and for N = 800, optimal combinations of β and λ are most likely to be discovered where λ is near to 0.1. It is reassuring that the landscape appears to be quite smooth at this resolution, implying that it should be a relatively easy matter to discover combinations of β and λ which give optimal performance.

The area which is suspected to hold optimal candidates for β and λ is next searched at a higher resolution, and the search is thus focused on a sub-set of the parameter space where reward achieved was on the optimal plateau noticed above.

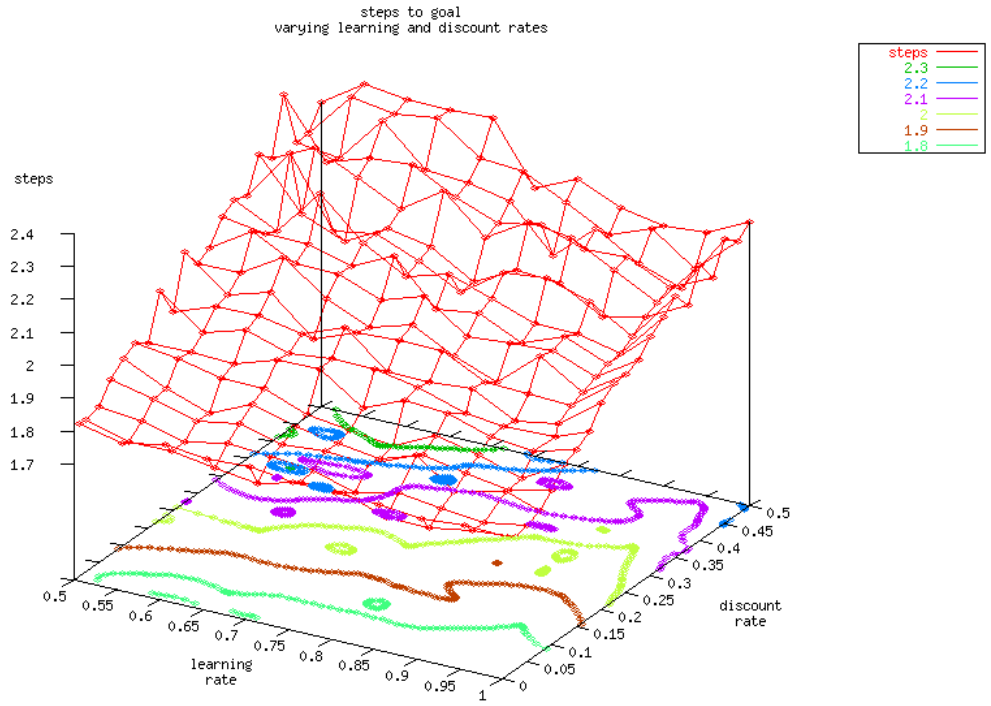


Figure 2-15 Homing in on optimal settings of learning rate and discount rate in Woods1e with stepwise reward and no cost of movement, w.r.t. average steps to goal

Once again from Figure 2.15, the same decrease towards optimal steps to goal is seen as discount rate is decreased. It would appear that the landscape is more rugged at this higher resolution, though it should be noted that the scale of the z axis is reduced. Clearly, the best solutions with respect to steps to goal are to be found where λ is less than 0.1.

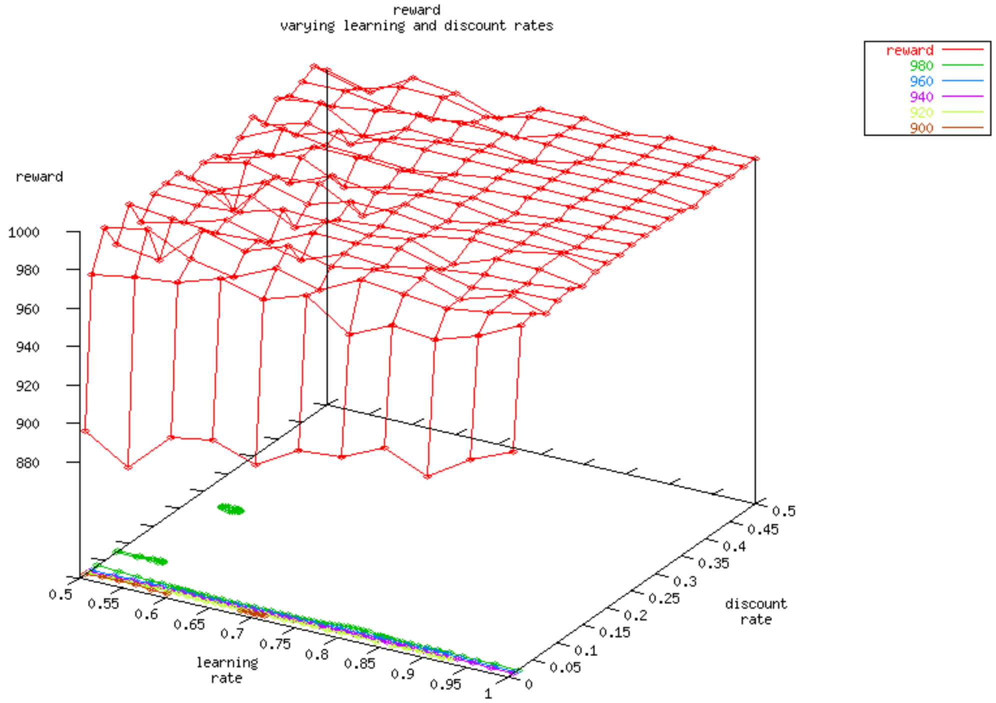


Figure 2-16 Average reward gained.

The same plateau is visible with respect to average reward gained (Figure 2.16). However, it is noteworthy that, in this more detailed investigation, there appears to be a rapid fall-off in reward as very low values of λ are approached. Once again, high values of β would appear to be advantageous.

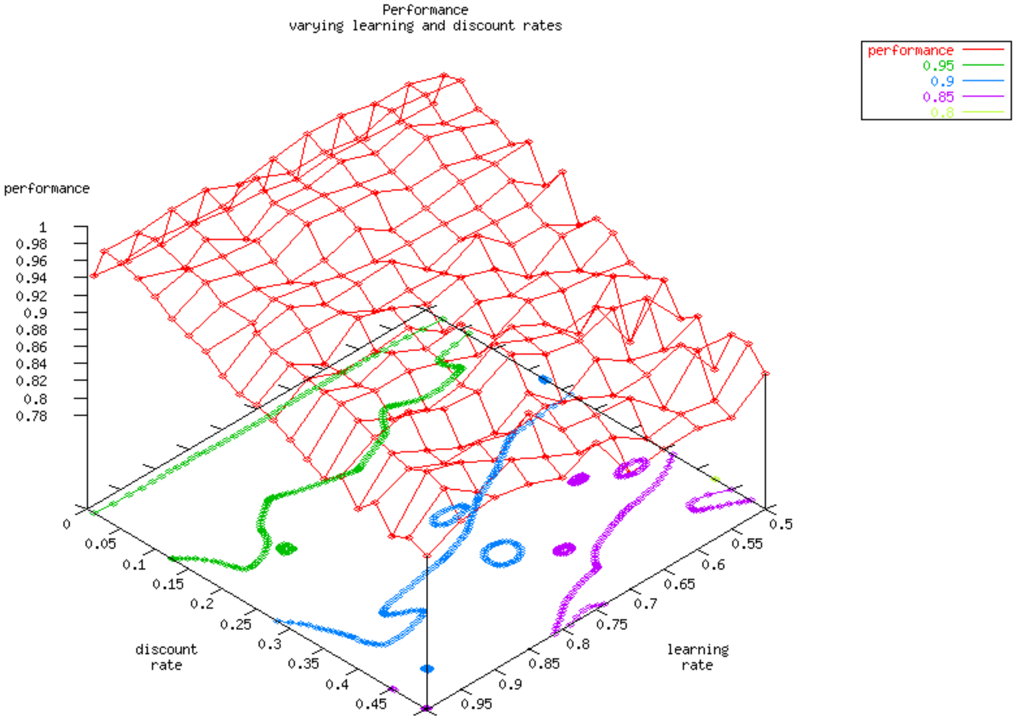


Figure 2-17 Performance.

The fall off in reward at the lowest values of λ causes a decrease in the overall performance of the system, as shown in Figure 2.17. From this graph it appears that a promising area for further investigation lies where β varies between 0.5 and 0.95, and λ is around 0.05. These results are presented in Figures 2.18, 2.19, and 2.20.

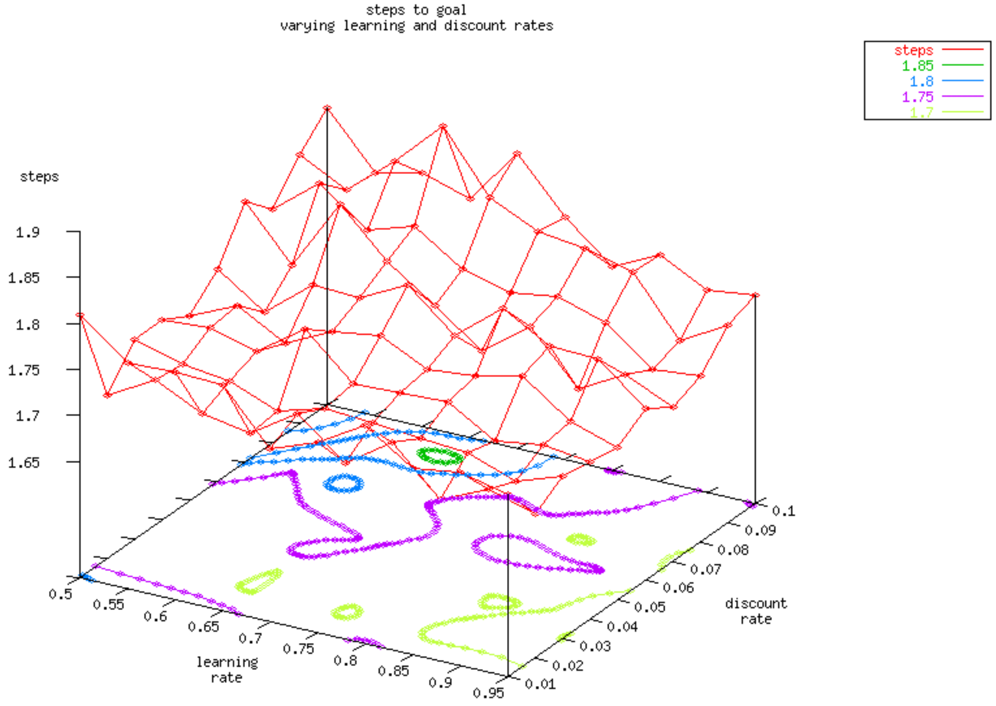


Figure 2-18 Optimal steps achieved

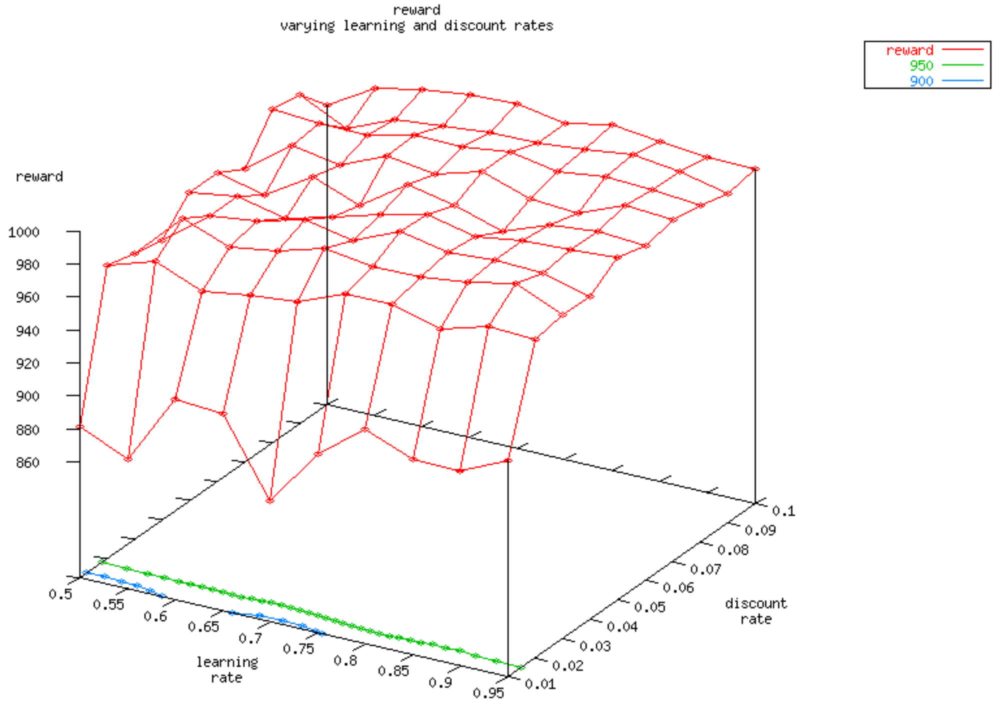


Figure 2-19 Reward.

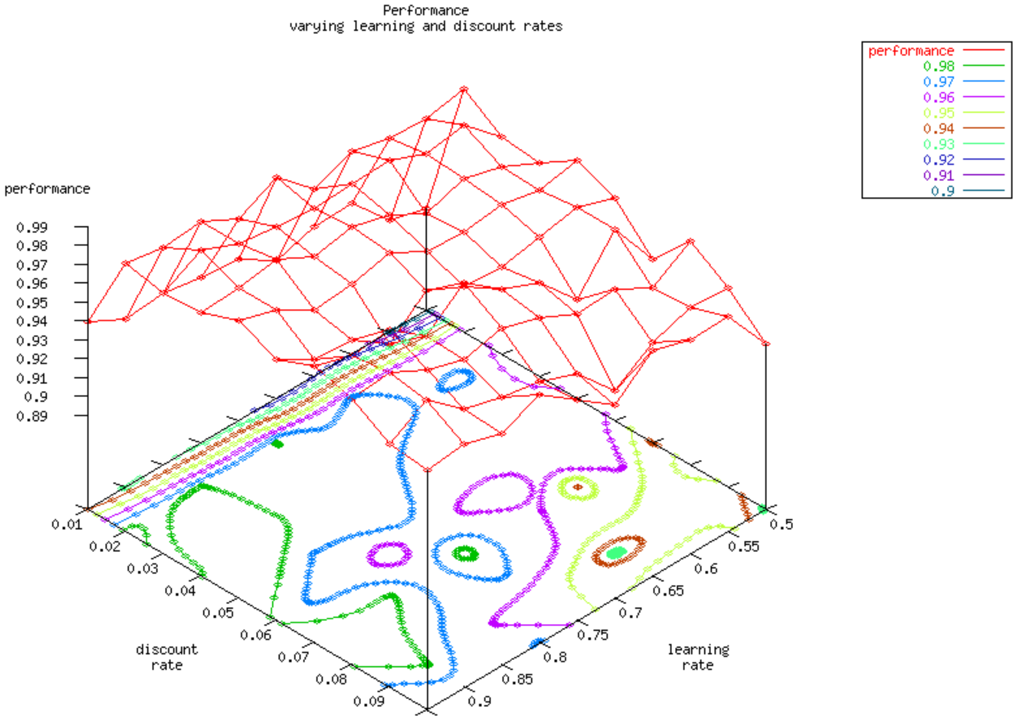


Figure 2-20 A ridge of near optimal performance, with peaks.

Using this method of successive refinement of the search of the parameter space, it is possible to find the combinations of β and λ which produce the best performance for a problem, with all other parameters fixed. As mentioned, it is reassuring that points in the parameter space seem seem to resemble more closely their near neighbours than they do more distant points, which underpins hope for success through this process of stepwise refinement.

As mentioned earlier, the other parameters that have found to be of importance for ZCS to attain optimal performance are population size, and the rate of genetic churn. As a simple exploration of how ZCS responds to these factors, we will briefly investigate the parameter space of population size against mutation rate. With $P_{\#}$=0.33, S_0=20, β=0.85, γ=0.05, τ=0.2, χ=0.5, ϕ=0.5 and ρ=1, so that the GA is triggered on every cycle, different values of N and μ,are explored (see Figures 2.21 and 2.22).

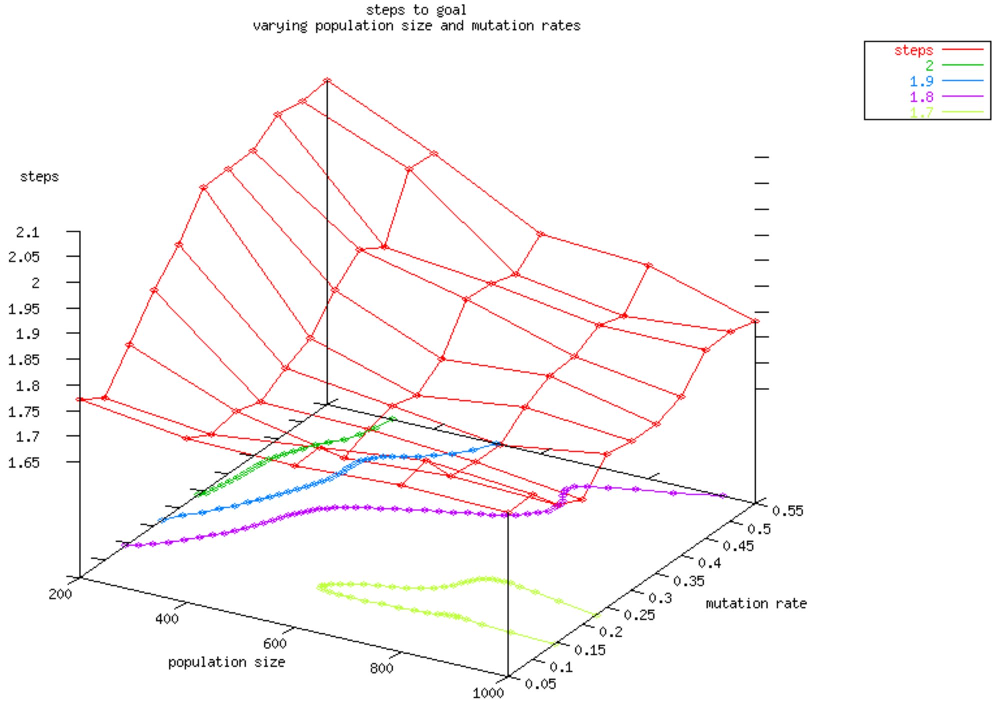


Figure 2-21 Steps to goal in Woods1e with stepwise reward and no movement cost, with varying population size and mutation rate.

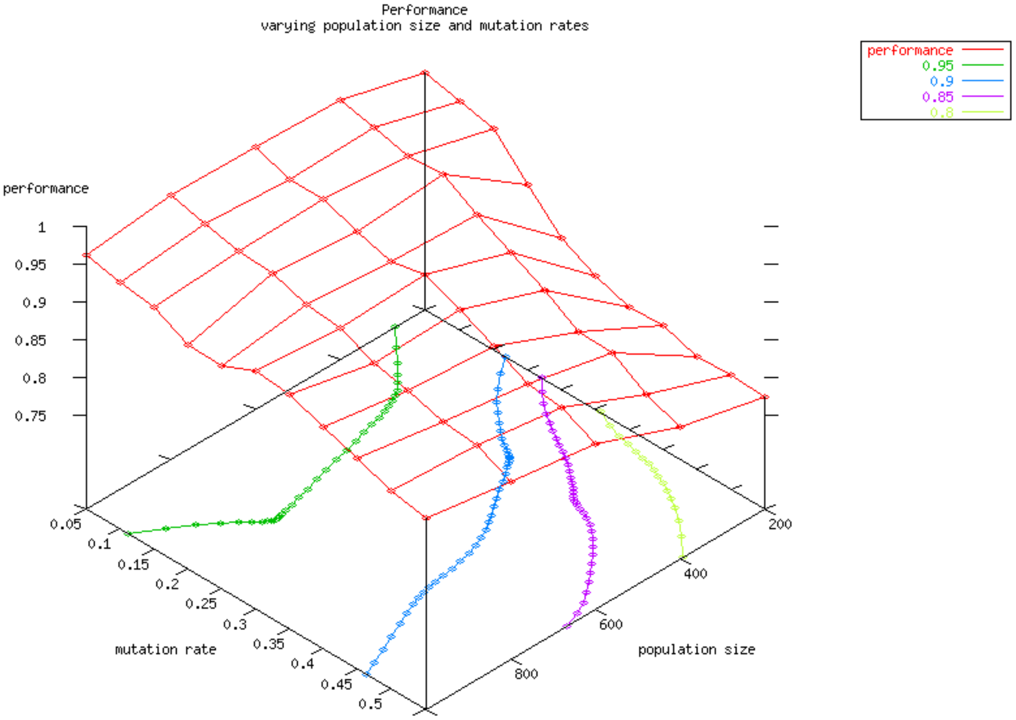


Figure 2-22 Performance with varying mutation rate and population size

Once again, it is useful to note that the landscape is smooth. Predictably, a higher mutation rate causes more deleterious effects with smaller populations than in larger ones. Differences in performance due to changing population size are less pronounced with lower mutation rates. This is presumably because there is a dilution effect with larger population sizes.

2.9 Moving towards real multi-objective problems.

The experiments above were carried out in the 'Woods1e' environment where the reward received upon reaching either goal state is proportional to the animat's 'energy level', and when there is no cost associated with movement.

Clearly, such a scenario is unrealistic. We shall now examine the performance of ZCS in the Woods1e environment where the animat's energy level is affected by the movements it makes, and where the reward it receives varies in direct proportion to its energy level. As before, an additional 20% extra trials are performed at the end of an experiment in 'deterministic mode' to show the underlying best performance achieved by ZCS.

2.9.1 Dynamic Cost, Stepwise Reward (woods1e-type2).

A cost of 0.01 'energy points' is now associated with each move made by the animat. This cost is deducted from the animat's internal energy level *after* it has been assessed whether the most recent action has brought an external reward, and if so, what size the reward should be. This is important; if the animat is charged for moving *before* the potential reward is assessed, the environment is rendered non—Markov in those trials where the internal energy level is 0.51. The animat moves to the 'food' state 'expecting' a high reward since the energy level is above 0.5, but this decision causes it to receive a low reward (see Figure 2-23 and Figure 2-24 below for a comparison of these treatments). All other details remain the same as in the case with no cost of movement.

As can be seen from Figure 2-23, ZCS proves itself to be capable of solving this problem optimally, with N=800, $P_{\#}$=0.33, S_0=20, β = 0.85, γ = 0.05, τ=0.2, χ=0.5, μ=0.002, ρ=0.25, ϕ=0.5.

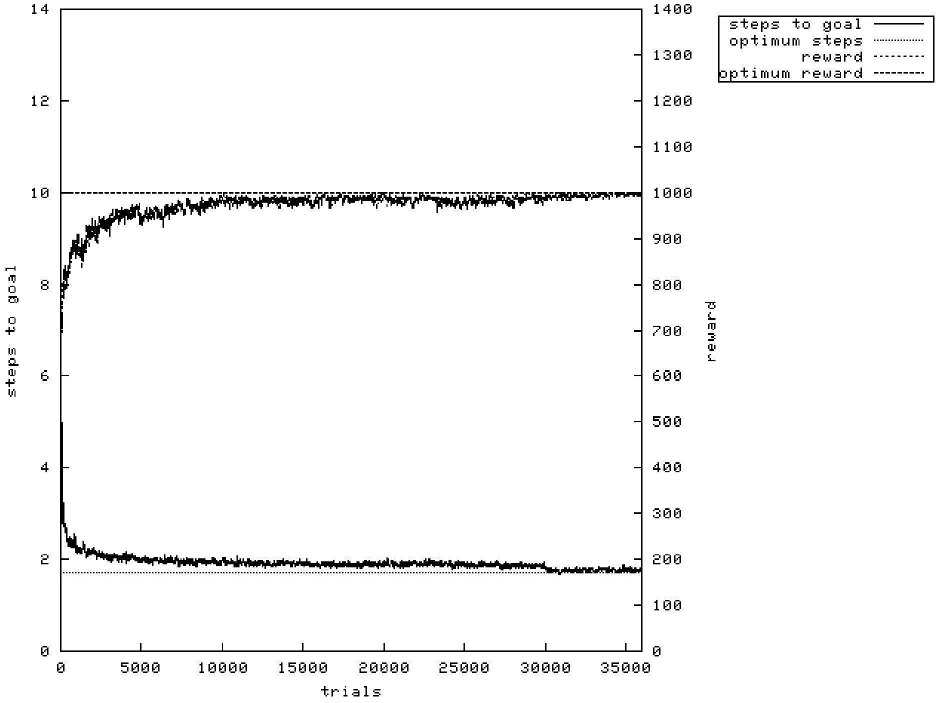


Figure 2-23 ZCS achieves optimal steps and reward in Woods1e with stepwise reward function and cost of movement

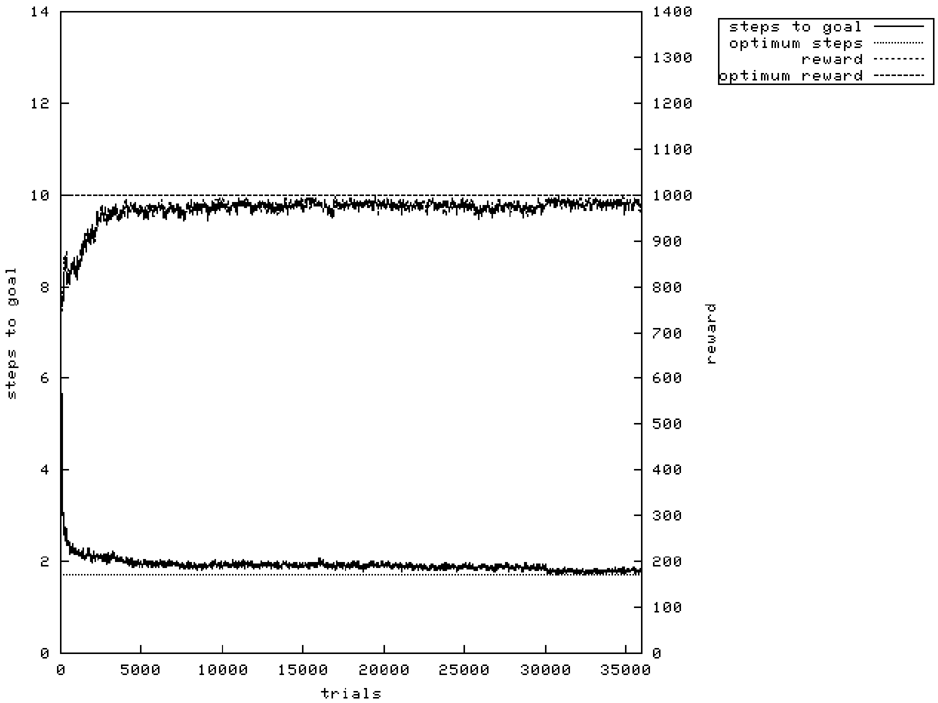


Figure 2-24 ZCS fails to achieve optimal reward with identical parameters when the cost of movement is imposed *before* assessing whether a reward has been gained.

2.9.2 Dynamic Cost, Continuous Reward (woods1e-type3).

In a further effort to make the simulation more like the problem faced by a real robot, ZCS is now set the task of finding an optimal solution to the Woods1e problem when its energy level is altered by its movements, and where the reward received upon arriving at one or other goal state is directly proportional to the energy level, rather than varying in a step-wise fashion according to the energy level as has hitherto been the case.

At the 'energy' goal	Reward = 1000.*e*
At the 'food' goal	Reward = 1000(1-*e*)

Where *e* is the animat's internal energy level which varies between zero and one. As before this internal energy level is set randomly at the start of each trial, such that approximately half the trials will start with e < 0.5. As in the previous experiment, a cost of 0.01 'energy points' is deducted for each move made by the animat in the grid-world. The animat's energy level is not allowed to go below zero. As in the preceding experiments, the real value of the internal energy level is hidden from ZCS, and is presented as an extra environmental character set to one if the energy level is above 0.5, otherwise zero.

While the optimum steps to goal remains 1.7 as in the other parallel multi-objective tasks in the Woods1e environment, the optimum average reward is no longer 1000 due to the dynamic reward. If the energy level at the goal state is above 0.5, achieving the correct ‘food’ goal will gain a reward in the range [1000, 500]. The same is true when the animat correctly reaches the ‘energy’ goal with its energy below 0.5. Assuming an equal random distribution of initial energy levels for trials, the average reward for success is 750.

As will be seen from Figure 2-25 and Figure 2-26, ZCS is capable of producing performance that approaches optimality, but a parameter set could not be found which produced both perfect reward, and also a perfectly parsimonious solution. This is discussed in section 2.10. The results presented in Figure 2-25 and Figure 2-26 were produced using the same settings as reported in 2.9.1 above, with the exception that in Figure 2-25 β=0.975, λ=0.775, and in Figure 2-26 β=0.98, λ=0.425.

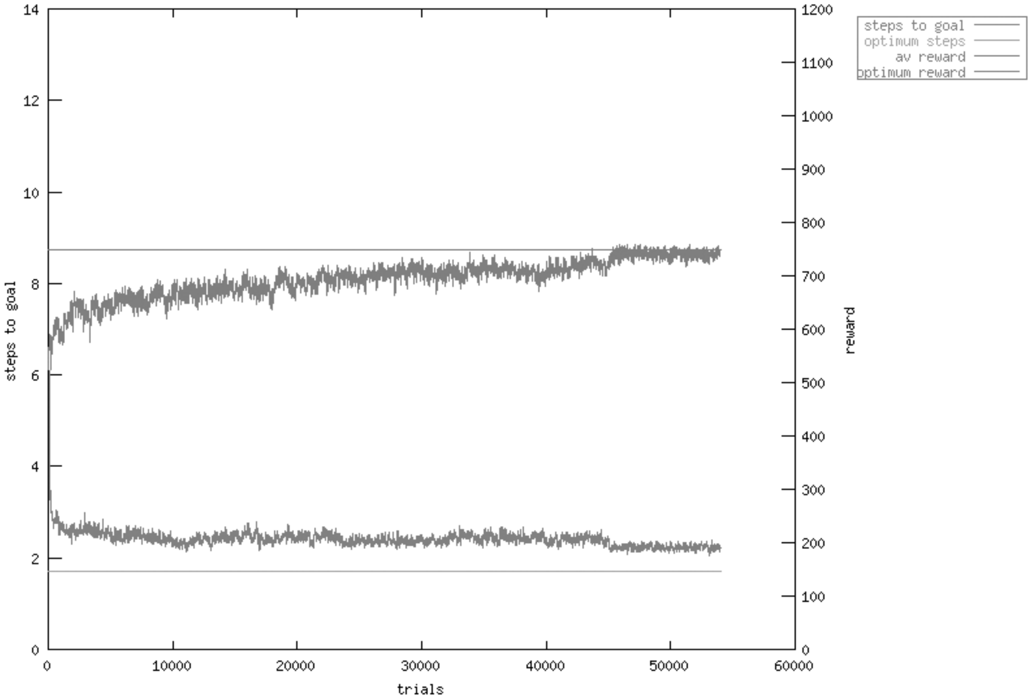


Figure 2-25 ZCS achieves optimal reward with sub-optimal steps.

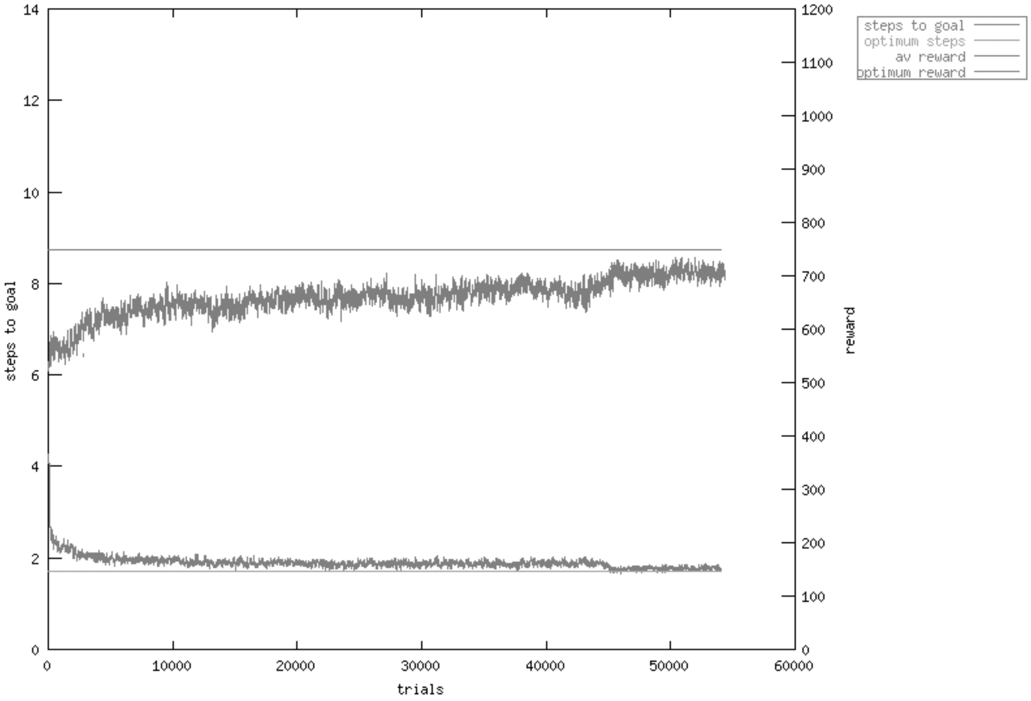


Figure 2-26 ZCS achieves optimal steps to goal with sub-optimal reward

It is important to underline the difference between this experiment and the preceding ones. Here the reward is dynamic and dependent on the learner's actions; the reward is not only dependent on deciding which goal to go to, but also upon achieving this goal in the minimum possible steps.

2.10 Exploration of the Parameter Space.

An exploration of the parameter space for the experiment with cost of movement and stepwise reward showed this surface to be very similar to that with no cost of movement. For that reason those results are not presented here. This is in contrast with the parameter space for the experiment in which there is a cost of movement and the reward varies in proportion to the internal energy level.

Finding a set of parameters which would allow ZCS to optimally solve this problem was a time-consuming process. Figure 2-27 through to Figure 2-35 show some part of this process.

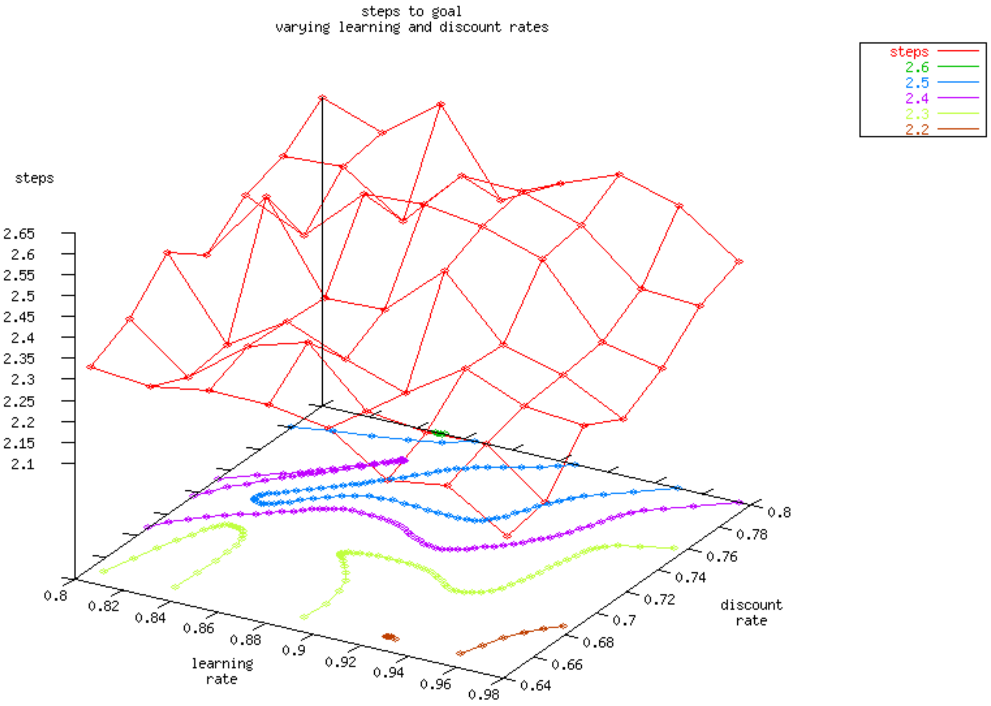


Figure 2-27 Search One : steps

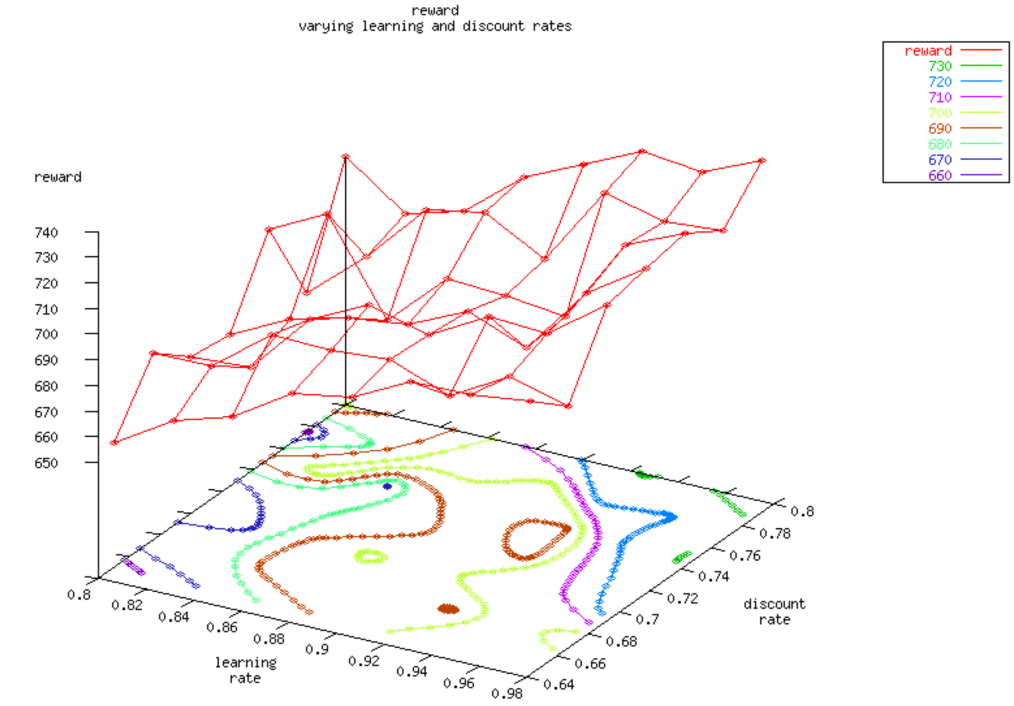


Figure 2-28 Search One : reward

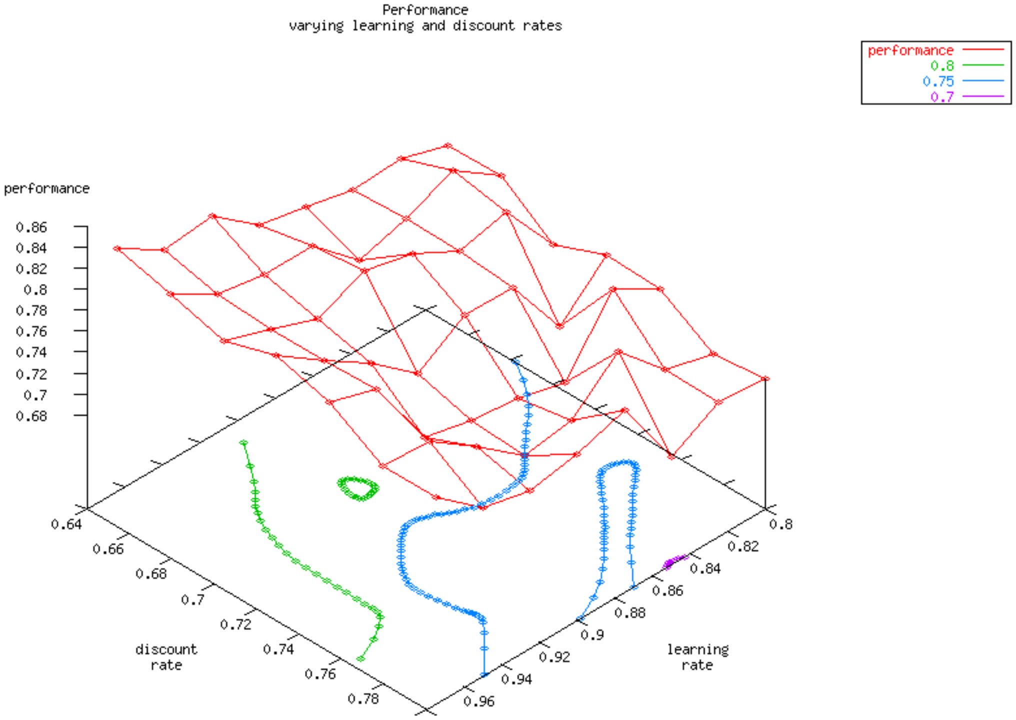


Figure 2-29 Search One : performance

Figure 2-27, Figure 2-28 and Figure 2-29 present the results of an initial scan of a sub-set of points in the space of β and λ. Earlier searches which led to this area are not presented for the sake of brevity, but had allowed the isolation of this region as a promising candidate for further investigation.

There appears to be an isolated region of good candidate solutions at approximately $\beta = 0.9$, $\lambda = 0.7$, and there is also an area where β is high where performance is good for a range of values of λ. The latter is investigated in the next search.

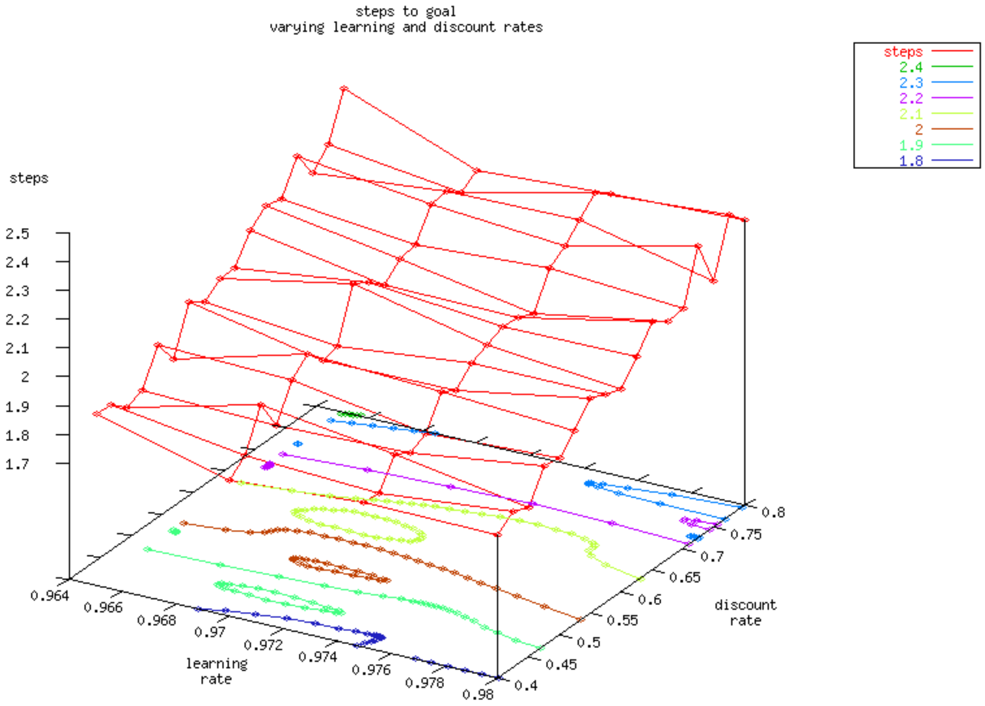


Figure 2-30 Search Two : steps

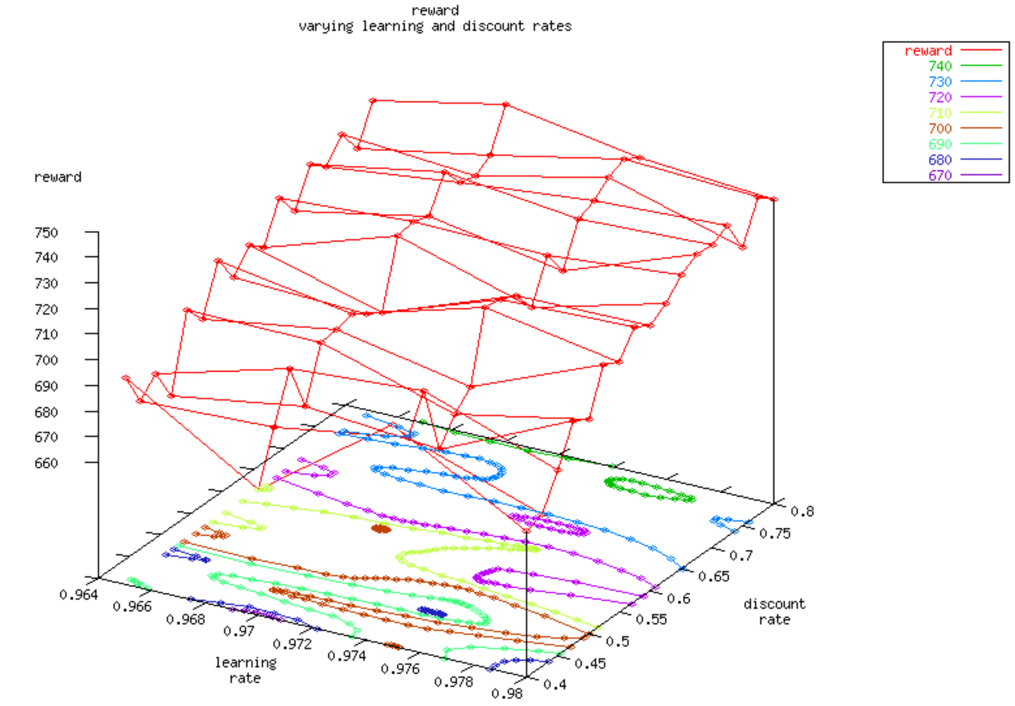


Figure 2-31 Search Two : reward

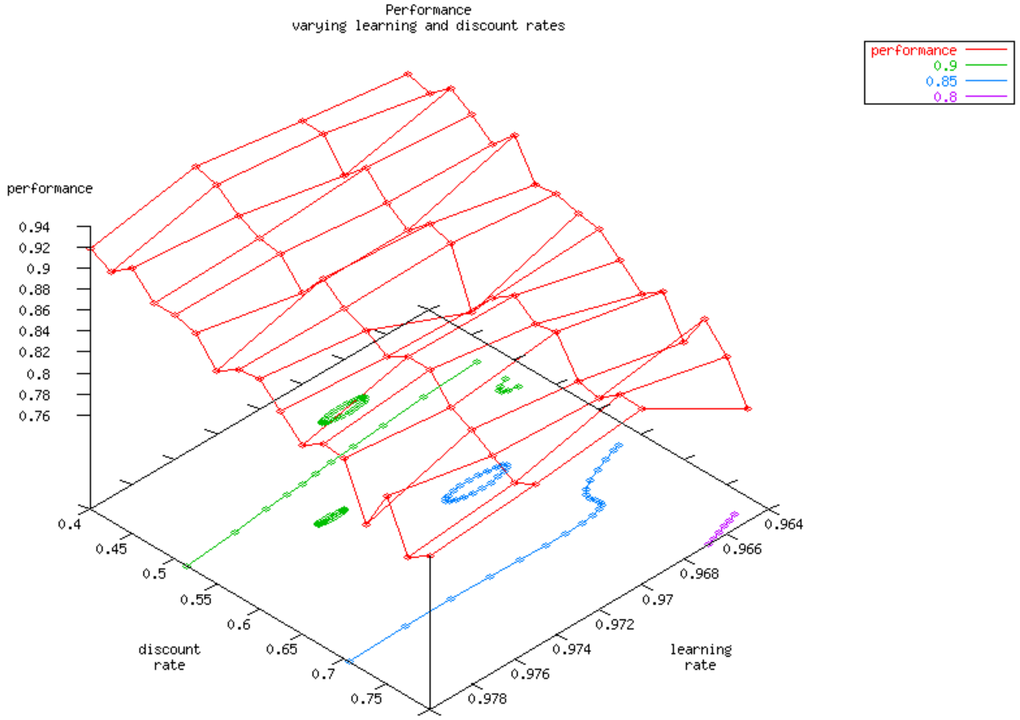


Figure 2-32 Search Two : performance

Figure 2-30, Figure 2-31 and Figure 2-32 explore this ridge of good candidate solutions noted in Figure 2-27, Figure 2-28 and Figure 2-29. It can be seen from these graphs that there is an unfortunate relationship between varying discount rate λ and steps to goal and performance. As λ decreases, steps to goal becomes more optimal, but the reverse is true for the average reward achieved. This makes it very difficult to find a parameter set which is optimal for both steps to goal and reward.

Figure 2-32 shows that the best compromise solutions exist in the region where discount rate is lower than 0.5, and learning rate is above 0.97. The search process now concentrates upon this region.

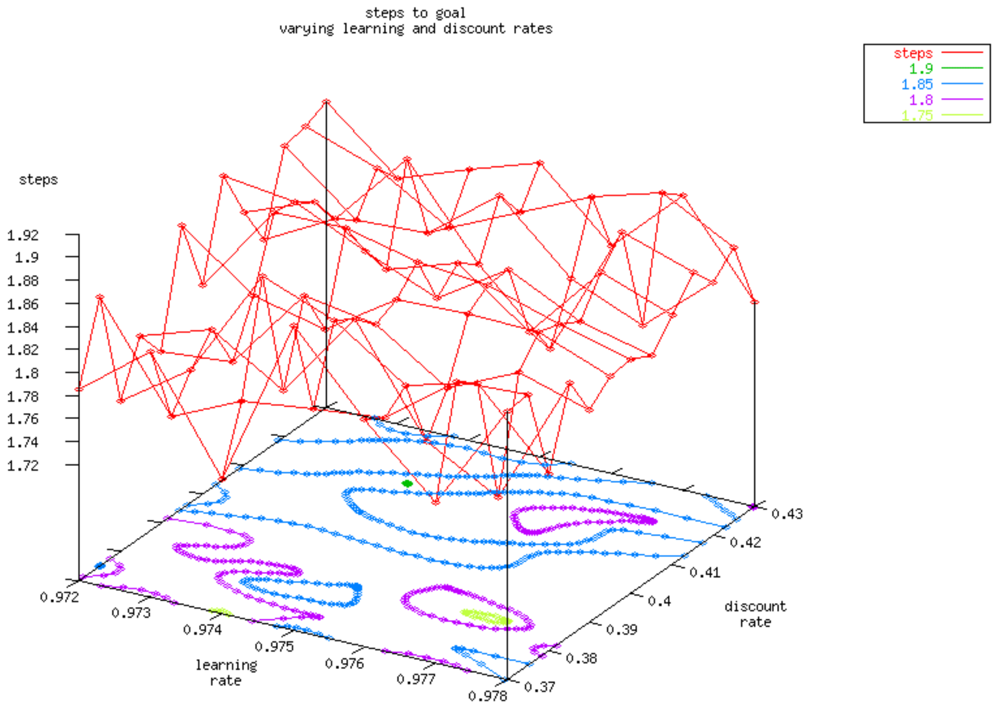


Figure 2-33 Search Three : steps

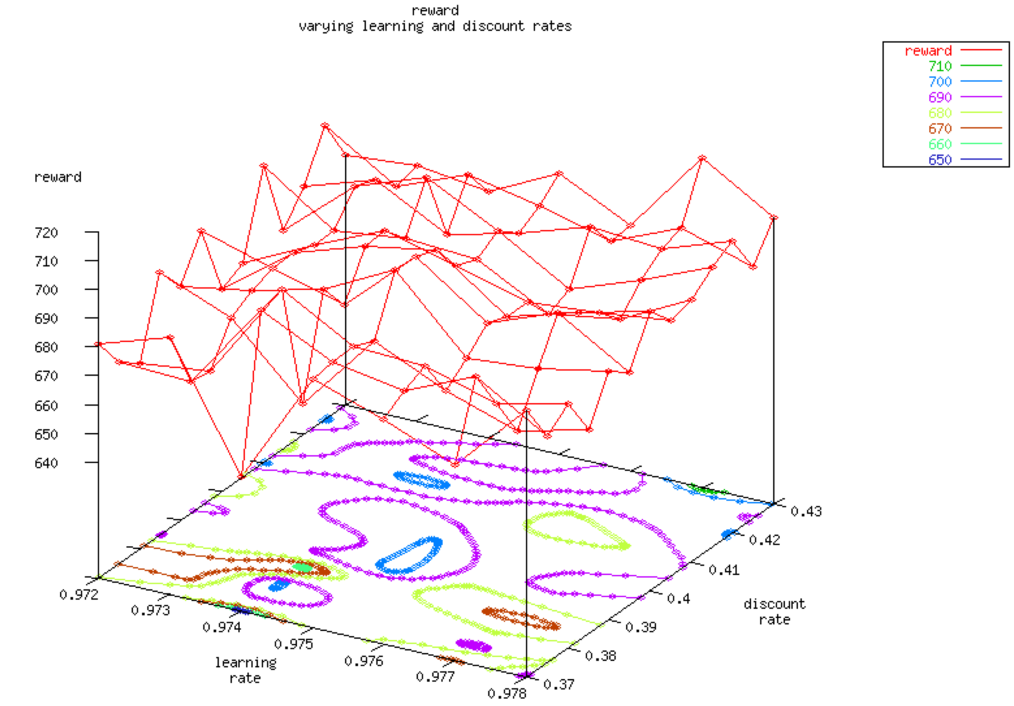


Figure 2-34 Search Three : reward

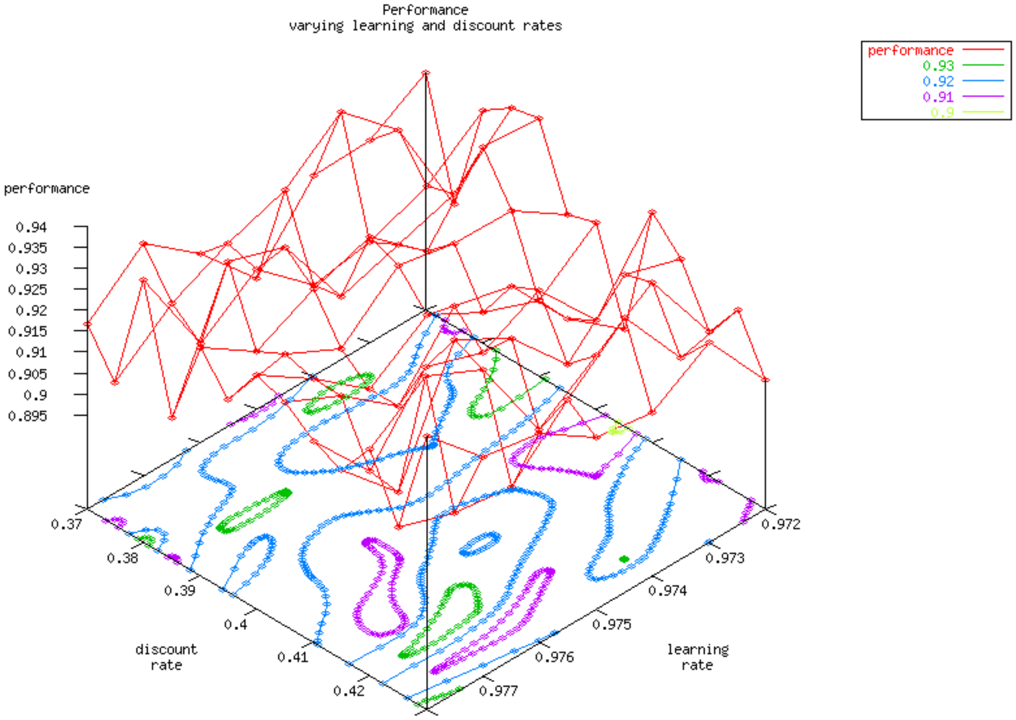


Figure 2-35 Search Three : performance

The surfaces formed in the graphs of steps, reward and performance in Search Three (as shown in Figure 2-33, Figure 2-34 and Figure 2-35) appears to be proportionally more rugged than they do in less detailed explorations. However, it should be noted that appearances may be deceptive – the scales on the Z axis are much reduced from those in the previous figures. Search Three does not clearly present a single region for further investigation.

As previously noted, there is a clear trade-off between optimal reward and optimal steps. Those points in the space of β and λ which gave high reward were poor in respect to the optimality of the number of steps to goal, and vice versa. Reward is dependent on two factors; going to the correct goal based upon the presented one or zero that shows whether internal energy is greater or less than 0.5, and the number of steps taken to get there. It would be possible to get the maximum reward by taking more and more steps until the internal energy level is zero, and then going to the energy goal. In other words, it is possible to deceptively get a better reward than could be achieved by finding the parsimonious and 'correct' solution. Solutions which take extra steps on their way to the energy goal will prosper.

Higher values of the discount factor λ give higher reward, but more steps to goal. Lower values of λ will mean that less reward flows down the implicit bucket brigade to states earlier in the chain. This explains the link between lowering the discount rate and decreasing the number of steps to goal. It may be that the link between higher discount rates and higher reward is explained by the fact that, with less pressure to reduce the number of steps taken, the possibility arises to 'cheat' in the way outlined above, taking more steps on 'energy trials'.

It is possible that a parameter set does exist which would allow ZCS to solve this problem in a fashion which is both optimally parsimonious and which gains an optimal reward. Having examined approximately 1250 unique sets of parameter settings, it must be stated that such a set is very difficult to find and at best solutions have been found which are nearly optimal for one or the other measure, but not for both.

2.11 More objectives.

Having explored the simple case of two objectives in the Woods1 environment, complexity is increased with three objectives. A third objective is introduced, termed ‘maintenance’.

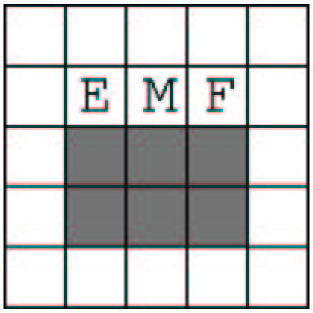


Figure 2-36 The 'Woods1em' 3 objective environment

As in the simplest 2-objective experiment, there is no cost of movement for the animat. The reward function changes slightly to accommodate the third objective.

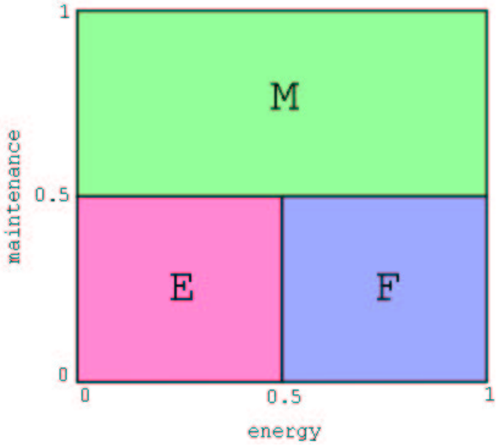


Figure 2-37 Graphical representation of correct goal in 3 objective Woods problem

If the animat reaches the ‘maintenance’ goal and its need for maintenance is higher than or equal to 0.5, it is rewarded 1000, otherwise 1, irrespective of the internal energy level. If the animat reaches the ‘food’ goal when the need for maintenance is lower than 0.5 and the energy level is higher than 0.5 it receives a high reward (1000), otherwise low (1). Conversely, if the animat reaches the ‘energy’ goal when the need for maintenance is lower than 0.5 and the energy level is lower than or equal to 0.5 it receives a high reward (1000), otherwise low (1).

All other experimental details remain as before, except that the condition part of the classifier is extended by one character which is set to zero if the animat’s maintenance level is lower than 0.5, and otherwise set to one. The optimum steps to goal[10] is approximately 1.8, and the optimum average reward that can be achieved is 1000.

All parameters are the same as in previous experiments, with the exception of the varying values of λ and β.

2.11.1 Results

Figure 2-38, Figure 2-39 and Figure 2-40 show steps to goal, reward gained and overall performance for an initial search of points across the space of discount rate and learning rate.

[10] Steps to goal is approximately 1.7 for the two previous goal states, but increases to about 2.1 for the new goal state, making the overall average approximately 1.8.

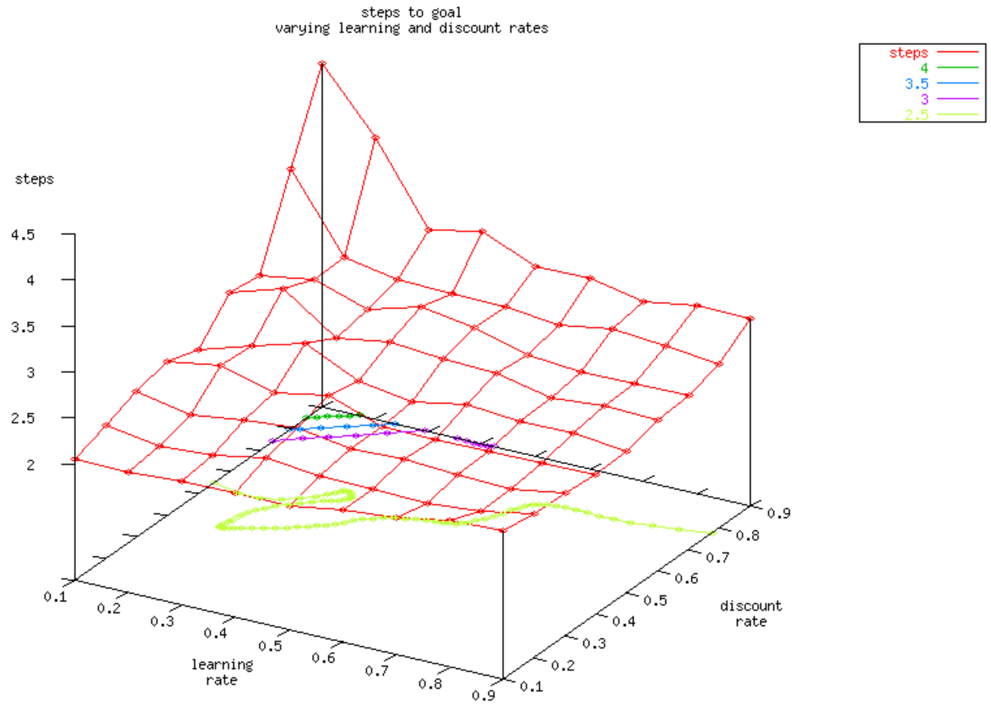


Figure 2-38 Steps to goal in 3-objective problem. Search One.

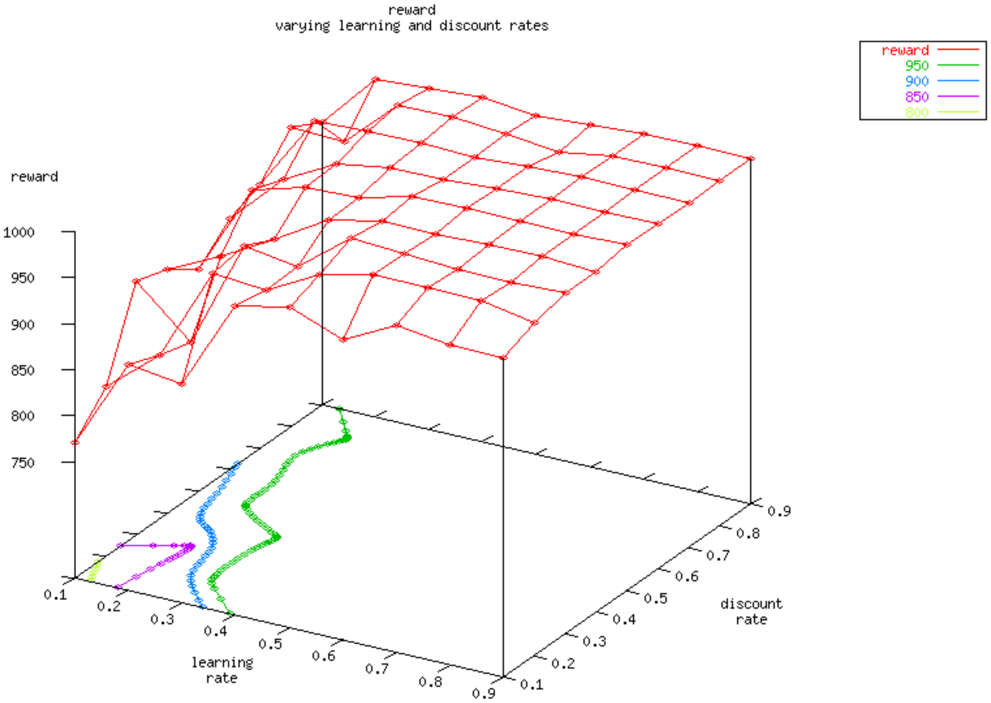


Figure 2-39 Reward in 3-objective problem. Search One.

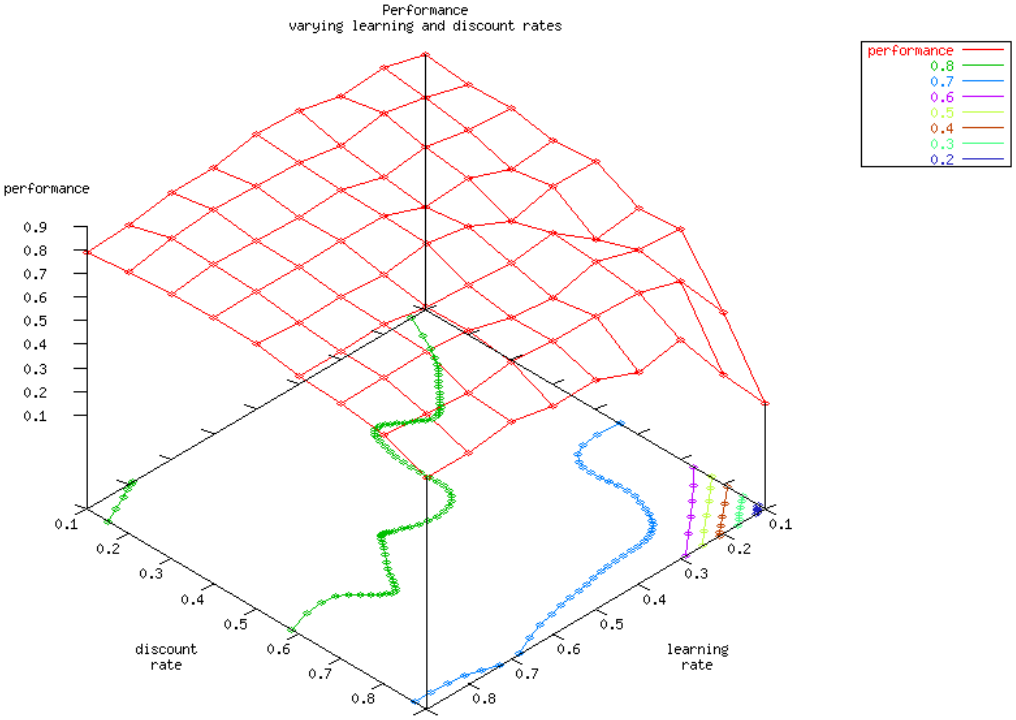


Figure 2-40 Performance in 3-objective problem. Search One.

As will be seen, once again there appears to be a large plateau in the space of β and λ values which produces solutions having optimal, or near optimal reward. Again, as expected the steps to goal fall as discount rate is lowered, imposing a greater pressure on parsimony in the solution. Even though the lowest value of λ used in search one was 0.1, it would appear from these figures that it is unlikely that optimal steps to goal can be achieved.

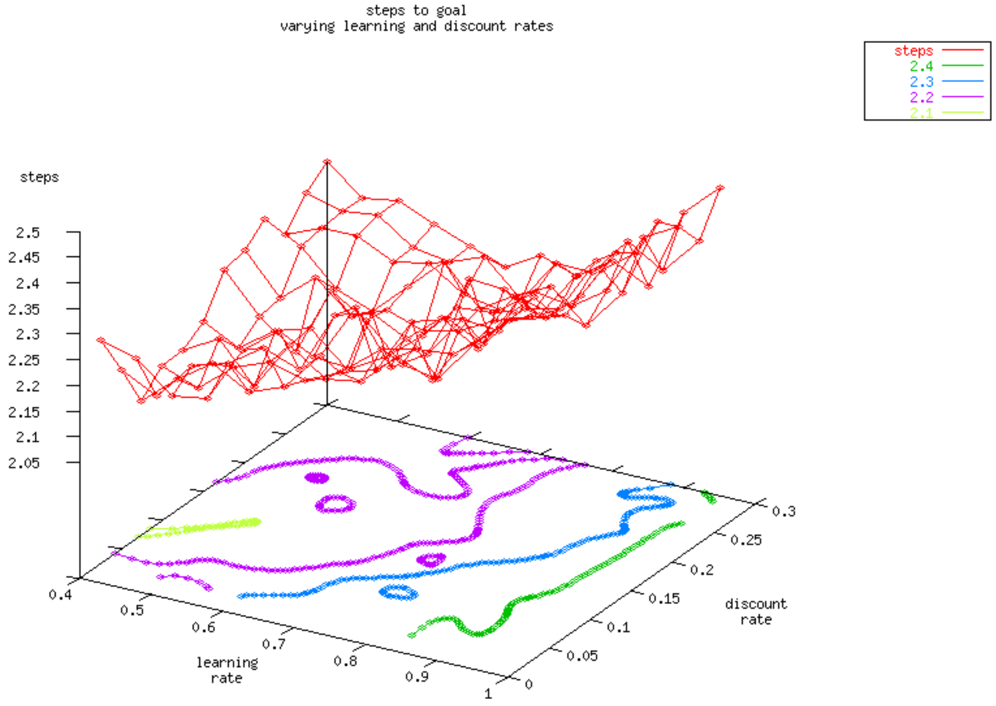


Figure 2-41 Steps in 3-objective problem. Search Two

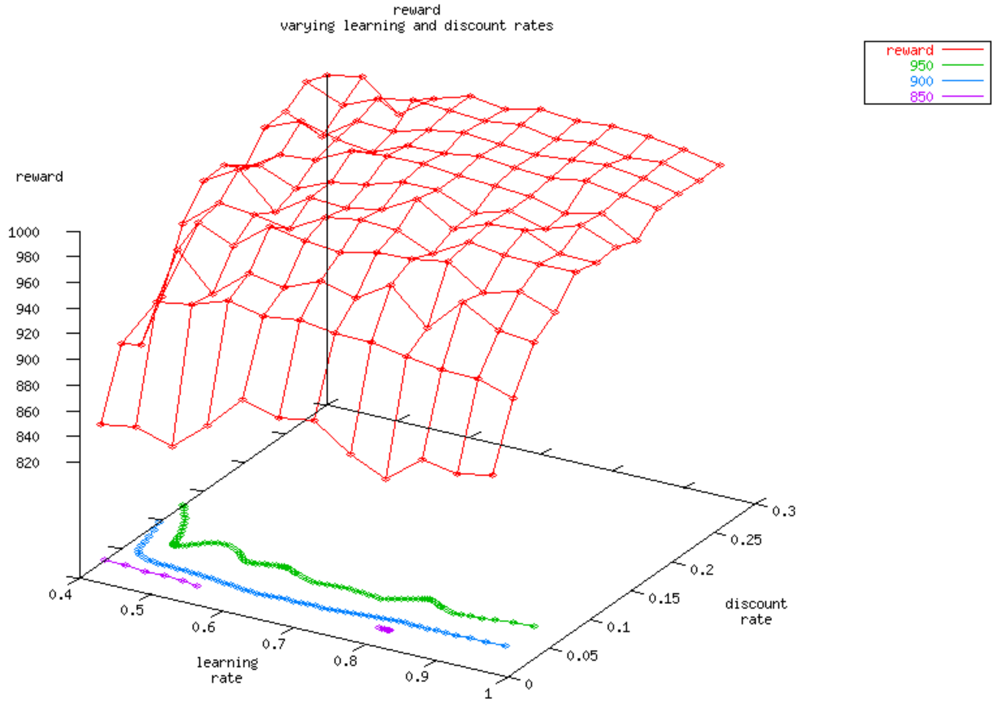


Figure 2-42 Reward in 3-objective problem. Search Two

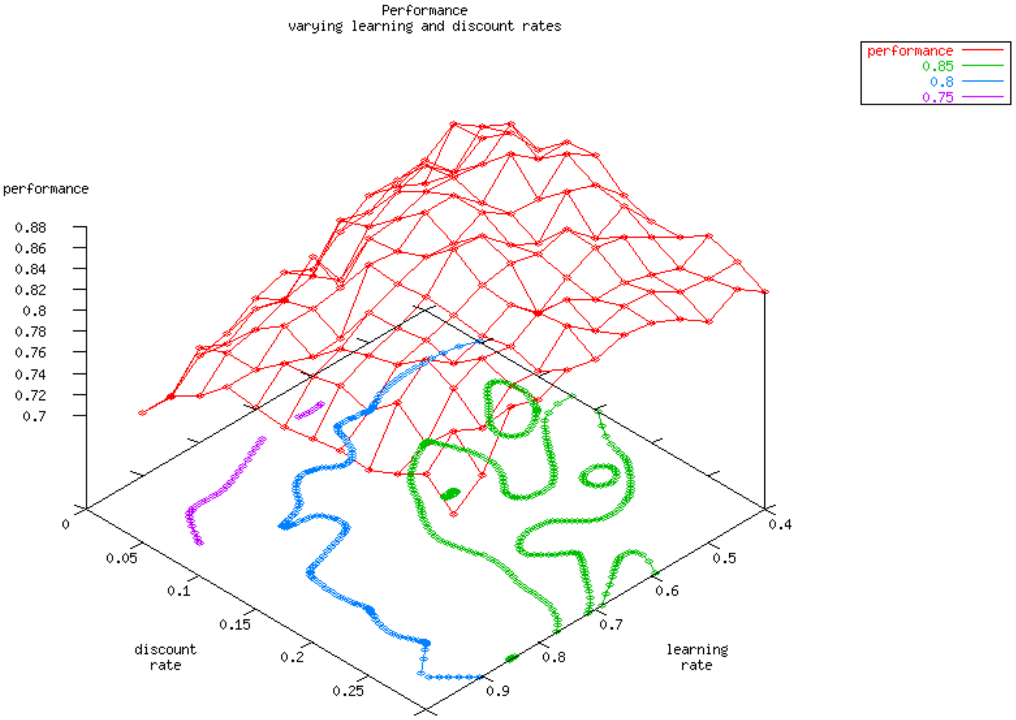


Figure 2-43 Performance in 3-objective problem. Search Two

Figure 2-41, Figure 2-42 and Figure 2-43 show a finer-grained exploration of ZCS's performance on the three objective problem in the region that appeared promising in search one. However, there are no good candidate parameter settings that suggest that further exploration would achieve an optimal solution. Steps to goal is still far from optimal. Further searches for an optimal solution by adjusting λ and β also failed.

It would appear that ZCS is unable to discover an optimal solution using a population of 800. It may be possible that it will achieve an optimal solution using more classifiers. Since there are few opportunities for generalization as discussed in Section 2, adding a second objective effectively doubles the space of condition-action pairs that must be explored by the classifier system, adding a third objective triples the size of the single objective problem, etc, so it is expected that the population required to achieve optimality to be approximately *o.N* where *o* is the number of objectives, and *N* is the size of population required to achieve optimality for one objective.

Some investigations of the parameter space for larger population sizes are reported below in Figure 2-44…Figure 2-49

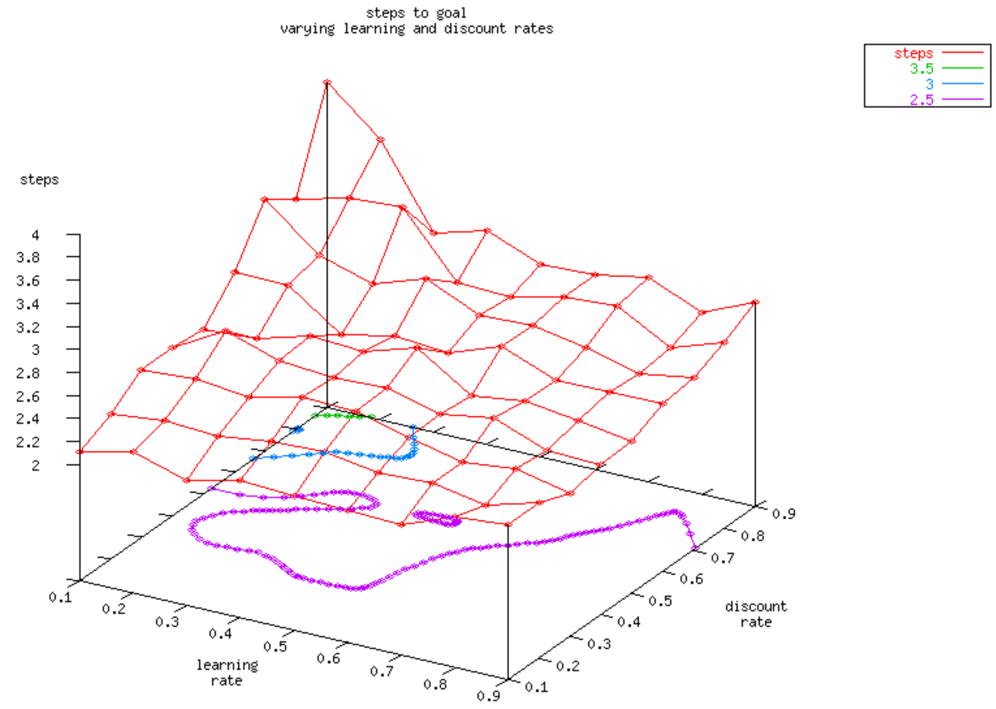


Figure 2-44 Steps to goal, *N*=1600

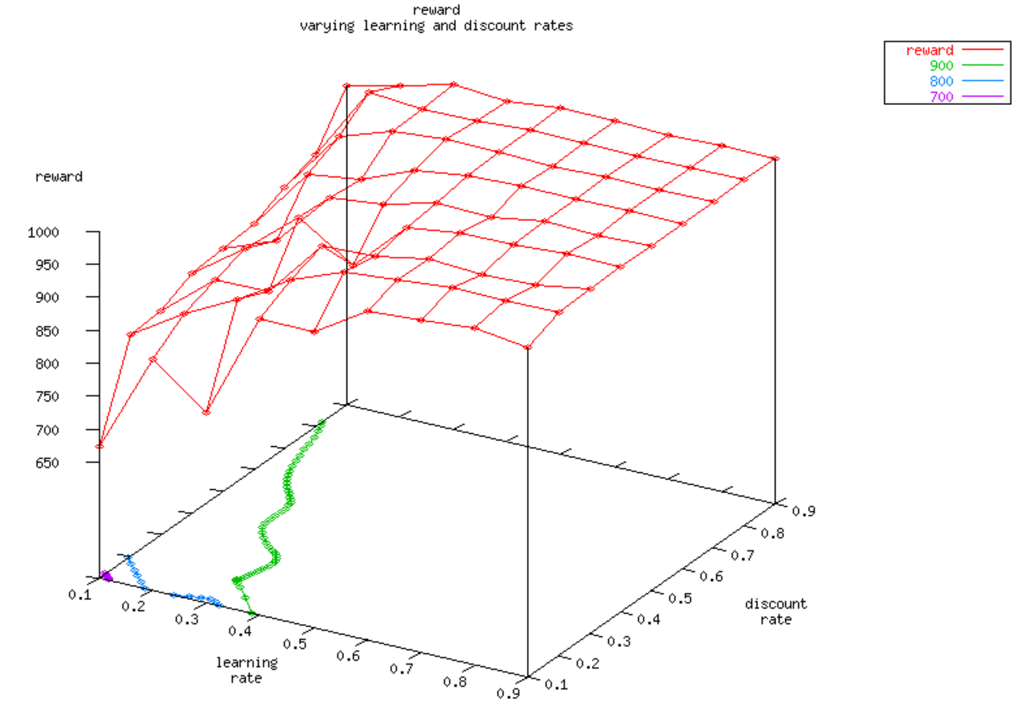


Figure 2-45 Reward, N=1600

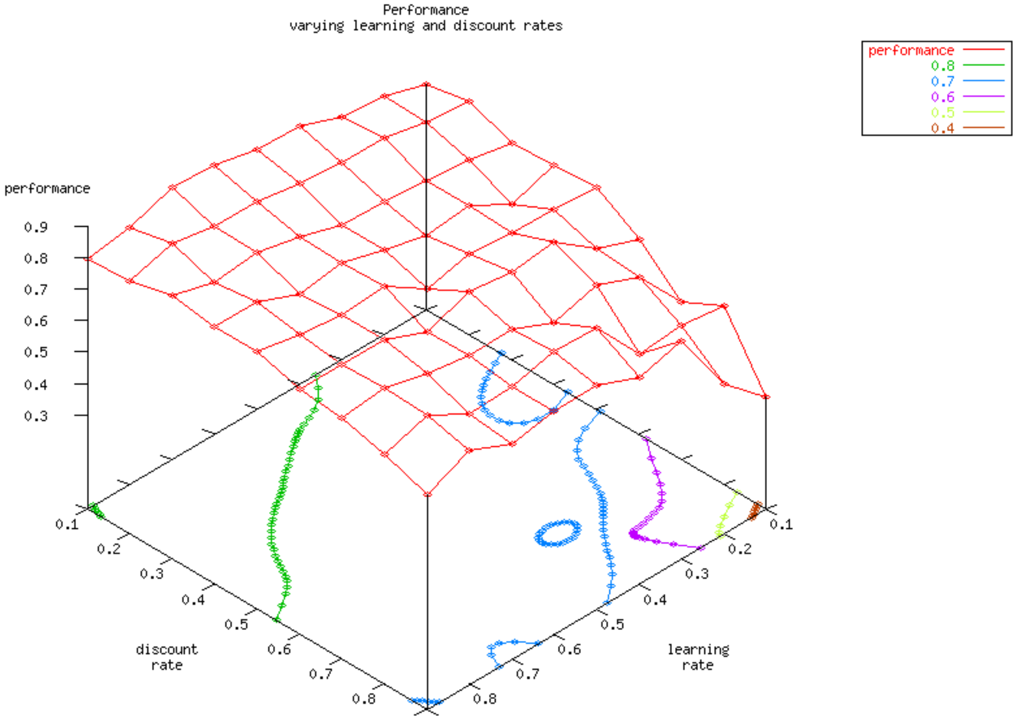


Figure 2-46 Performance, N=1600

Figure 2-44, Figure 2-45 and Figure 2-46 show results for N=1600. There is a slight improvement over the results achieved with N=800 (Figure 2-38, Figure 2-39 and Figure 2-40). It is interesting to note the similarity of these two sets of figures; increasing the population size effects small changes in the performance landscape, but the general topology remains the same. Thus areas which appear to hold the promise of good solutions with a small population may be expected to be equally promising with larger populations. This implies that the degree of epistatic linkage between population size and the reinforcement learning parameters is not as strong as the linkage between the reinforcement parameters themselves. Good settings of β and λ for small populations are likely to be good for larger populations.

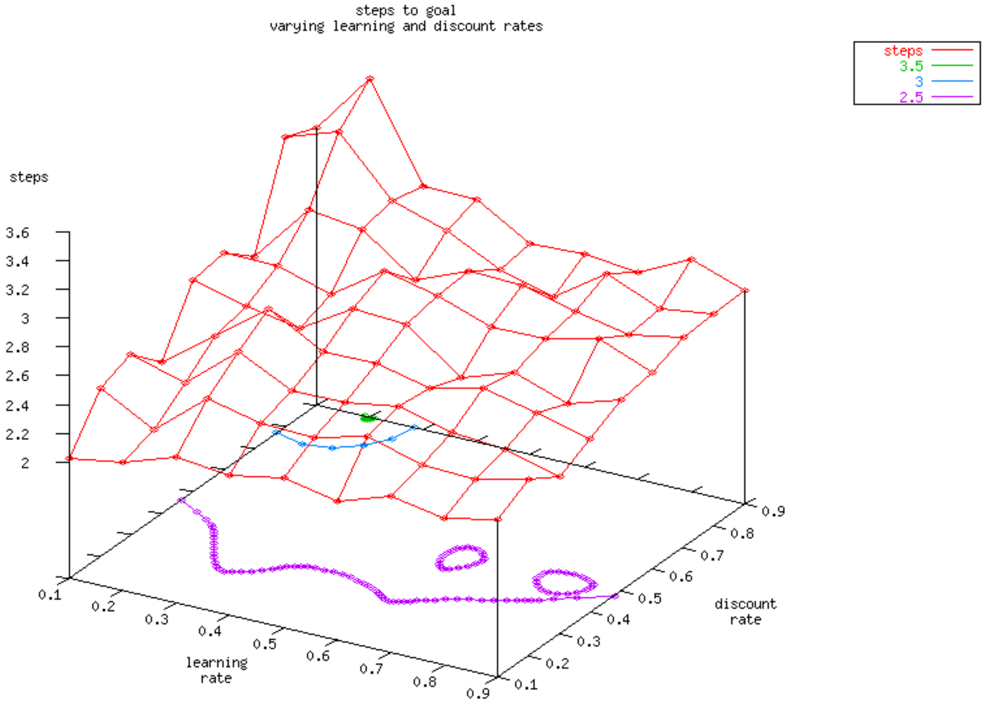


Figure 2-47 Steps to goal, *N*=3200

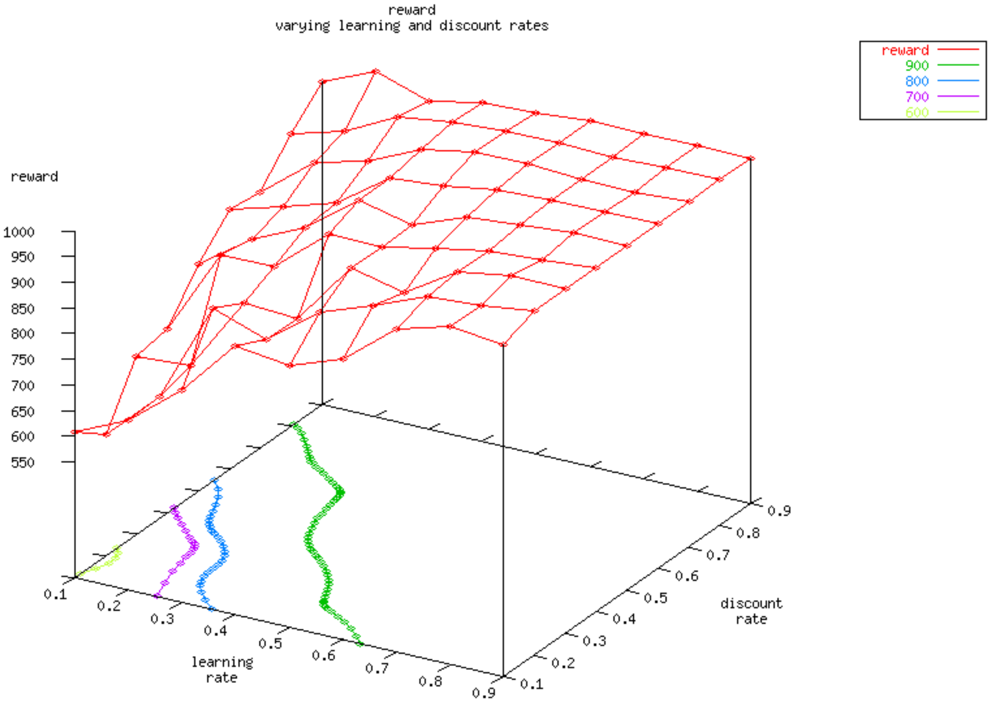


Figure 2-48 Reward, *N*=3200

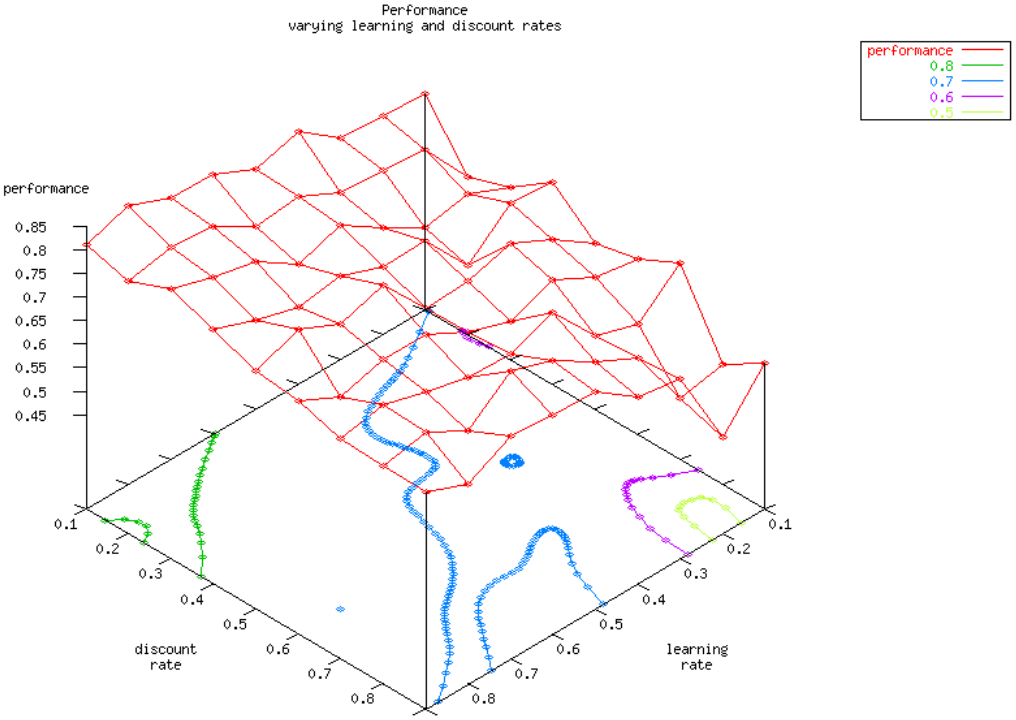


Figure 2-49 Performance, *N*=3200

Figure 2-47…Figure 2-49 show results for *N*=3200. Again, there is a slight improvement with the increase in population size. Again, the landscapes have a broadly similar topology, reinforcing the belief that an initial hunt for ‘sweet spots’ in the parameter space can be carried out using small populations, and these promising areas then explored more fully using larger populations.

It may be that optimal solutions could be discovered with these, or larger, populations. However, to find parameter settings which allow optimality is a time-consuming process. In order to use classifier systems in multi-objective problems with more than two objectives, the experimenter may be advised to ignore the simpler ZCS. In Chapter 4 further three-objective problems are examined using XCS, an accuracy-based classifier system which proves much more successful at approaching optimality with less tuning.

2.12 Conclusions

We have seen that ZCS can achieve an optimal solution in a simple multi-objective problem in which the actions of the learner change its environment; imposing a cost of movement may cause the ‘correct’ goal to switch during a trial as the animat’s energy level drops below a critical threshold.

The surfaces formed from measures of ZCS performance with different settings of discount rate and learning rate are in all cases quite smooth, and seem to tend towards being uni-modal. This suggests that in simple problems it should be fairly straightforward to converge upon a set of parameters which will allow an optimal solution.

We have also seen that an extension to this problem in which the reward is a function of the internal energy level is much more difficult to solve. The problem appears deceptive, allowing the learner to gain extra reward at the expense of taking further steps to goal. Although a great number of parameter settings were examined, a solution could not be found that was optimal both in respect of steps to goal and reward.

The complexity of the simplest task, with no cost of movement and a step-wise reward function, was increased by adding a further goal. This proved impossible to solve with the same population size that was successful for the two objective task. This is unsurprising. It appears that using ZCS to tackle these problems is unrealistic, since with the harder problems tens of thousands of experiments had to be performed to find performance that was near to optimal.

Although ZCS can therefore produce optimal behaviour in some simple multi-objective problems, the process of refinement of parameter settings is hardly the basis for a useful on-line learner, since it requires the assessment of many sub-optimal solutions before the parameter settings can be adjusted. While some parameters can be self-adapted effectively in ZCS [Bull2000b], they could not demonstrate optimal performance through the self-adaptation of β and λ.

Chapter 3 TCS : Multi-objective robot control

3.1 Introduction

In the previous chapter, ZCS was shown to be capable of performing some simple multi-objective tasks. These tasks took the form of various grid-worlds, which provided a scalar reinforcement signal when an animat, the movements of which were controlled by ZCS in response to a binary representation of its environment, arrived at one of the various goals. In this chapter, an adaptation of ZCS is used to address similar multi-objective problems on an autonomous robot.

The task of controlling a physical agent in the real world presents more complicated challenges. Unlike a grid-world, the real world is a continuous environment. At each iteration of ZCS's cycle of sense, decision, action, and reinforcement, the environment is presented to the population of classifiers, and an action must be chosen. In a continuous environment, by analogy, the controlling classifier system must be presented with an encoding of the environment, and the robot moved accordingly. How far should the robot move? With no 'grid' in the world, how often should the world be presented to the classifiers, thus driving the reinforcement process?

One approach would be to allow the robot to move in fixed increments. The controlling classifier system waits for the movement to be completed, and the cycle continues. This has the advantage of simplicity; the continuous environment has been reduced to a grid-world. However, it raises its own problems. If the fixed-sized increments are too big, it may be that a problem cannot be solved optimally, or at all. For example, consider a robot moving in one metre increments in an environment in which parts are less than a metre in size, or in a situation where it is only half a metre from the goal. Conversely, if the fixed-sized increments are too small, the solution may be inefficient. For example, consider a robot moving in one centimetre steps in an environment where it is 10 metres from its goal; a single 10 metre move cannot be made, and the robot is forced to take 1000 small steps to get to the goal. Clearly, in order that a learner can solve a problem in the real world in a fashion that is both efficient and optimal, the size of the incremental moves - and therefore the resolution at which the environment is experienced - must be adjusted to the characteristics of the environment in question.

Since the resolution of the actions must be adjusted to the environment in which the learner finds itself, it would be advantageous if an appropriate resolution could be discovered and if this discovery could be a function of the learner itself. Furthermore, the learner should be able to take big or small actions in the same environment, as appropriate. For example, in the task of driving from home to a friend's house in a city on the other side of the country, a large scale map of the motorways may serve for most of the journey, but towards the end one requires a smaller scale map of the destination city and its minor roads.

Interlinked with the problem of determining the appropriate scale of movement is the problem of how fast the cycle of sense, decision, action, reinforcement should be performed. If the cycle is paused while an action is taken, then the learner is oblivious to any important events that occur during this time. This would clearly be a handicap in any but the most trivial autonomous application. Any real robot must maintain the ability to interrupt the movement it has attempted – it must respond to unforeseen events resulting from an incomplete understanding of the world, or changes in the world in order that it can avoid potholes and bumping into experimenters' legs.

This chapter presents an overview of some related work in which learning algorithms are applied to robotic control problems. It shall be shown that in many cases learning is a slow process requiring significant *a priori* decisions to be taken by the experimenter. An adaptation of the ZCS classifier system is then described in which the scale of movement appropriate to the environment is automatically determined. This algorithm is implemented on a robot platform, and is tested in both single and dual objective real-world problems.

3.2 Related Work

3.2.1 Evolutionary Robotics.

There have been many attempts to apply evolutionary algorithms such as Holland's Genetic Algorithms, Rechenberg's Evolution Strategies, Koza's Genetic Programming, and Evolutionary Programming to the problems of generating control programs for robots. Most often, the approach used is that of Genetic Algorithms [Walker2003].

As described in Chapter 1, the approach to developing robotic controllers using simulated evolution by GAs is to create a population of solutions, which each either directly or indirectly encode a controller for a robot. Each of these controllers is then tested to evaluate its utility, or fitness. Since the assessment of fitness is the most time-consuming step, requiring a physical robot to interact with its environment in what may at first be a quite random fashion, the evolutionary robotics approach is hampered by the fact that all members of a population must have their fitness assessed for each generational cycle of the algorithm, and a number of such cycles may be required in order to converge upon even the simplest controller. For this reason, much work that purports to describe the evolution of controllers for robots is done only in simulation.

However, there have been some noteworthy examples of the use of evolution on real robots. In most, simulation is used as a short-cut; after training in simulation, the controller is moved to a real robot to test the evolved system. For example, Jakobi et al. [Jakobi1995] developed a simulation of the popular ‘Khepera’ robot. The Khepera has eight IR sensors that also receive some signals in the visible light spectrum. The simulation was based upon an idealized mathematical model of the sensors, environment, and kinematics of the Khepera. A GA was used to evolve neural network controllers in simulation, and these were validated on the physical robot. Controllers were evolved for some simple problems including light-seeking and obstacle avoidance.

Jakobi [ibid.] also experimented with the introduction of noise into simulations. Perhaps unsurprisingly, he showed that when the level of noise in the simulation is similar to that of the real system, evolution in simulation can produce controllers that work well on the physical robot. The controller experiences problems when the amount of noise in simulation is too low or too high; when the simulation is inaccurate the evolved controllers will not transfer easily to the real robot.

Nolfi et al. [Nolfi1994] built a simulator of a Khepera, though theirs was based on actual recorded sensor data. They too used this simulation to evolve neural network controllers, and then transferred to the physical robot to affirm the validity of the evolved solution. They found that the controller evolved in simulation performed less well on the real robot, but interestingly noticed that a few iterations of evolution on the real platform adjusted for this, showing that the simulation had been quite close to reality.

There are obvious problems with this approach. As Brooks famously noted; '*The world is its own best model*' [Brooks1986], thereby issuing the rallying cry for behaviour-based robotics in which behaviour is orchestrated by the interaction of independent but interlinked modules, with no concept of a symbolic internal model of the world. Evolutionary robotics, perhaps developing artificial neural control systems, is clearly allied to the sub-symbolic behaviourist school. However, to build a model of the world in which to develop a controller seems self-defeating. Rather than have the learner build a model, we have built the model ourselves. The simulation will most likely only be applicable to a small set of the problem domains in which we might like to evolve controllers, and therefore the costly process of building a sufficiently accurate model must be undertaken again and again.

Also, it may be that there are some classes of robot task we can envisage in which a robot has to perform a task in an environment which we cannot accurately model or predict. For example, consider a space-exploration mission. We may define the purpose of the mission – perhaps to collect rock samples – but we cannot know how the environment will be perceived before the robot arrives. If our simulation of the unknown is too inaccurate, we cannot expect the evolved controller to function as we hope.

Thirdly, there are two goals in the automatic production of robot controllers by adaptive means. One goal is the production of a controller for a robot without the need to write this control program ourselves. This goal could be addressed by evolution in a suitable simulation. The second goal is that of generating a controller that is itself adaptive – it has the capacity to learn during its lifetime. In order to address this second goal of 'lifetime learning', adaptation must take place on the physical platform even though it is possible that an earlier training phase could usefully have been in simulation.

While it may seem merely a pragmatic approach to shorten the lengthy training process, the use of simulation in evolutionary robotics is therefore a contentious issue. There are some examples of work in which evolution of robot controllers has been carried out solely on robot platforms. Some notable examples are described below.

Cliff et al. [Cliff1993] report the evolution of vision systems. Initial experiments were done in simulation, and latterly on a gantry robot in which a suspended camera could be moved in two dimensions. An angled mirror below the camera reflects the image of what is ahead of the robot. This visual input is pre-processed on a workstation before being presented to the recurrent artificial neural networks that are the phenotypic realisation of the evolved genotypes. Each individual in the population consisted of two chromosomes, one specifying the position and size of three receptive fields within the image, and the other the architecture of a neural network. Values averaged from pixels in the receptive fields are then presented to the input nodes of the neural network, as are bumper information. Artificial evolution was carried out in three stages of increasing behavioural complexity, with a modified environment and fitness function being used for each stage. Using this methodology they demonstrate the evolution of a controller that will approach a white triangle attached to the arena wall, and avoid a white rectangle.

Floreano and Mondada [Floreano1996] also used a GA to evolve a population of strings of floating-point numbers representing weights and threshold values of a discrete time recurrent neural network topology. These controllers were tested on a Khepera, where the inputs to the network were the values of the IR sensors, and the outputs were velocity commands sent to the two motors of the Khepera's wheels. The authors describe two sets of experiments, in both of which the environment was carefully controlled (for example, there was a single constant source of light in the room containing the experimental set-up). In the first the robot learnt to walk along a winding corridor without hitting the walls. The evolution process took at least 50 generations until the best individuals were approaching optimality, and each generation took about 40 minutes to complete, a total of about 33 hours until good behaviour had been discovered. In their second experiments, in which evolution continued for 10 days, the fitness function depended on the distance travelled by the robot – in order to travel far, the simulated battery which discharged in 20 seconds would have to be topped up by recharging, which the robot could do by driving over an area of the arena which had a different floor colour (the floor colour could be detected using a downwards-pointing sensor).

Matellan et al. evolved fuzzy controllers for a Khepera, again to perform the navigation and obstacle avoidance tasks [Matellan1998]. Similarly to the work of Floreano and Mondada, the genomes that coded for the fuzzy controllers were evolved on a workstation, and each individual controller was embodied on the Khepera in order to assess its fitness. There was a clear improvement as the evolutionary process continued. With a population of 100 individuals being tested for 20 seconds each on the Khepera, over 100 generations, the total time spent in evolving the controllers was 55 hours, excluding failures and accidents.

Hornby et al. describe a series of experiments to evolve gaits for the Sony AIBO ‘robot dog’ [Hornby1999, Hornby 2000]. In their earlier experiments they evolved gaits for the AIBO, but found that these were not necessarily robust enough, experiencing difficulties when the AIBO was on a different surface to the one on which the gait was evolved, or when a controller evolved on one AIBO was installed on another. The locomotion of the AIBO is controlled by a central locomotion module, that produces different gaits through the setting of 61 parameters. For the evolutionary experiments, all except 20 were set to constants. Experiments were performed on two types of surface, one smooth and the other ridged. The parameters of the locomotion module are set to each of the evolved sets, and the AIBO then measures how far it travels in seven seconds. The authors suggest that the technique worked well in that it both developed robust controllers, developed novel, interesting gaits, and automated a tedious manual process. Each run of 500 generations took around 25 hours to complete.

Marocco and Floreano used an evolutionary approach to evolve an active vision system for a Koala robot [Marocco2002]. Active vision is the use of movement to shift objects in the visual field, making it possible to recognise or disambiguate them. A GA was used to evolve a population of genomes encoding weights and thresholds in a fixed-topology recurrent neural network. The fitness of the evolved controllers was assessed on the task of driving around an arena without bumping into the walls, based on the input of a pan-and tilt camera that returned a 240 x 240 grid of black and white image information. Each of 25 input nodes encodes a value from a non-overlapping 48x48 tile of the image. Two additional input nodes encode camera pan and tilt information. Output to effectors was by two nodes which delivered real-valued motor commands for forward or backwards velocity, and by two nodes that similarly set the pan and tilt motors of the camera. Each generation took 1.5 hours, and after eight generations (12 hours) the best and average performance had been reached, although evolution was continued until the 15th generation (22.5 hours). The reader is referred to the book by Nolfi and Floreano for an overview of evolutionary robotics, and further details [Nolfi2000].

In summary, evolution has been shown to be an effective way of developing simple robotic controllers. For example, when the genotype is used to specify the weights for a neural architecture, the direct input from sensors can be used without mandatory pre-processing. However, evolution requires the assessment of the fitness of many phenotypes. If this requires the phenotypes to be embodied, the process is slow. Speed-up can be achieved by the use of simulation, but this runs counter to the philosophical and pragmatic reasons for evolving the controllers.

3.2.2 Reinforcement Learning on Robots.

Reinforcement learning consists of iteratively learning better estimations of the optimal *value function* that predicts the best reward that can be achieved in a particular state by taking a particular action, and that the optimal policy is followed thereafter. In environments with continuous state or action spaces, the value of an infinite number of state/action combinations must be stored. This would make learning very difficult, since the learner is unlikely to encounter any particular situation again. The learner must therefore generalise in some way. This presents a major challenge to the use of reinforcement learning in continuous environments.

A number of researchers have used reinforcement learning on robotic platforms and a few of the more noteworthy are mentioned below. Mahadevan and Connell [Mahadevan1991] used Q-learning to control a robot named ‘Obelix’. The robot’s task was to push boxes across a room. Obelix was based on a Pioneer robot with a sensory system consisting of eight sonar units, each with a field of view of 20 degrees, of which four look forward, two point to the left and two to the right. Each of these sensors can return one of two values, ‘NEAR’ and ‘FAR’. An Infra-Red (IR) sensor on the front of the robot is switched on when an obstacle is pressed against the front of the robot, allowing the recognition of the state termed ‘BUMP’. The electric current supplying the motors used for forward motion is also monitored. If this exceeds a threshold value, the robot is determined to be ‘STUCK’. The environment is thus presented in 18 state bits, comprising two possible value for each of the sonar sensors, and one each for ‘BUMP’ and ‘STUCK’. These 18 bits provide 2^{18} perceptual states; approximately 250, 000.

Motor control outputs for the robot were limited to five choices; moving forwards, turning left or right by 22 degrees, and turning left or right by 45 degrees.

The robot's task was to learn the mapping between perceptual states and the five motor actions such that it could best push boxes around the room.

In initial experiments they attempted to use Q-Learning to develop a monolithic controller that would encode the state-action mapping. However, this was abandoned in favour of a modular approach. It was discovered that when reinforcement was based upon 'a simple reward function', reward was obtained too infrequently, but when a more complicated reward schedule was devised the robot became trapped in local minima, for example, trying to avoid everything in the room, or to push everything. Also, perceptual aliasing was a problem – even with its quarter of a million distinguishable states, the robot was unable to tell the difference between some areas of the environment, '*boxes often looked like walls in sonar images*'.

Mahadevan and Connell then split the architecture of the robot into three behavioural modules, applying a 'divide and conquer' methodology. The behavioural modules which would be linked together in the fashion of Connell's 'colony-style' architecture[11] [Connell1990] were;

- 'Finder'. Rewarded when the input contained 'NEAR' bits, and punished when 'NEAR' bits previously on are turned off. This behaviour was intended to move the robot towards boxes, and was the lowest level behaviour in the subsumption stack.

[11] Also termed 'winner takes all' subsumption, i.e. the actions of the robot are those suggested by a single module.

- ‘Pusher’. This module was intended to learn how to push boxes. A reward was given when ‘BUMP’ was on and the robot continued to move forward, with a negative reinforcement applied when this became no longer true (the robot had been pushing a box but had then lost it). This behaviour was switched on by the ‘BUMP’ bit.
- ‘Unwedger’. At the top of the subsumption stack, this module would start if the ‘STUCK’ bit is switched on. The robot would receive a reward if the ‘STUCK’ bit was switched off, and a negative reinforcement should it remain switched on. This module was intended to remove the robot once it had pushed a box into an immovable object, or if the robot became stuck.

Mahadevan and Connell state that the robot was ‘fairly successful’ in learning these behavioural modules. The overall behaviour after learning was said to be close to that achieved by the hand-coded agent.

It is important to note the role of *a priori* knowledge of the problem in this work. Without a hierarchical decomposition of the problem into separate modules, Q-learning was unable to find an adequate solution. Also, the environment was pre-processed and sonar signals were classified into the categories ‘NEAR’ and ‘FAR’. Since the authors report some perceptual aliasing, this imposed ‘discretisation’ clearly hides potentially important environmental information from the learner. However, it is the discretisation which provides the generalisations that enable Q-learning to be practical in this example.

A similar approach of generalising through the *a priori* discretisation of continuous data is used by Asada et al., who have published many works in the field of robot soccer. In [Asada96] they tackle what seems to be a much more complex task than the one approached by Mahadevan and Connell, namely, to use Q-learning to solve the problem of shooting a ball into a goal using video camera input. Once again, the environment is simplified into discrete categories before being presented to the learner. The ball's position is classified in terms of its position as right, centre, or left, and in terms of its size and hence distance as large/near, middle, small/far, i.e. 9 states. The goal's position is also quantified in terms of distance and relative position, and in addition in terms of relative angle (angled away to the left, to the right, or straight-on), i.e. 27 states. Finally, two states exist for when the ball has left the visual field to left or right, and likewise when the robot loses sight of the goal. The environment can therefore be described in a total of 9*27*2*2 states = 972.

The authors do not impose a predetermined hierarchy on the method of solution, and also do not present complex pre-programmed actions in the action set. The actions available to the robot are three commands, forward, back, and stop, that can be sent to each of the two motors, making a total of 9 actions.

The authors used a sparse reward function, giving a positive reinforcement of '*1 when the ball is kicked into the goal and 0 otherwise*'. Once an action has been chosen, the robot continues to take that action until the environmental state changes, at which point another action is chosen and the action value function is updated. In this way an action always causes the transition from one state to another, even though the real distance travelled in order to achieve this transition might be different in different parts of the environment – for example, in order to cause the perceived angle of the goal to change, the robot need move less far when it is near to the goal than when it is further away.

Mahadevan and Connell experienced problems with developing their 'monolithic' version of Obelix when using a sparse reward function (the delayed reinforcement problem), and thus tried complex reward schedules which resulted in convergence on local minima. It was for this reason that they adopted an *a priori* decomposition of the task into learnable modules. In contrast, Asada et al. [Asada1996] increase the frequency of reinforcement by using a methodology they term 'Learning from Easy Missions' (LEM). Put simply, the robot is initially started in states from which it might be expected easily to achieve reinforcement, and thereafter placed into successively more difficult conditions. By using this methodology they demonstrate impressive results on this difficult problem. They note that in comparison with a hand-coded fuzzy logic system, the robot learner is less efficient. They suggest that this might be due to the learner having achieved reward in some cases after taking the wrong actions, such that '*the optimal path obtained by the learning method might include detours*' [Asada1996]. It might be the case that optimality would have been achieved had the experiment continued for longer.

In both the work reported by Mahadevan and Connell, and Asada et al, the world has been discretised according to an arbitrary scheme invented by the experimenters. This approach, while effective in allowing the learner to build a table of values mapping discrete states to actions, limits the learner's ability to deal with different environments. The discretisation may need to be changed (by the experimenter according to his specialist understanding of the new environment and learner) to learn the same task in a bigger room. If the discretisation is too coarse-grained then optimality cannot be achieved, but since Q-values must be established for every combination of state and action a fine-grained discretisation will result in having to build a huge map of state-action values, thus impeding learning. The memory requirements grow, and inefficient use is made of experience – since each state-action value must be estimated in isolation, no use is made of the fact that states near to each other are likely to have similar values and similarly optimal actions [Kaelbling1996].

In order to address the problem of the combinatorial explosion of state-action values that must be learned, *function approximators* [Sutton1998] are widely used to generalize over the state-action space. Many techniques have been applied, including neural networks, fuzzy logic, and CMAC [Santamaria1998].

Santamaria et al. demonstrate the use of CMAC, or Cerebellar Model Articulation Controller, in which each input activates some subset of overlapping tiles of the state-action space. The predicted Q-value is the sum of the values represented by these ‘features’. Clearly, the size of the ‘tiles’ controls the generalising abilities, and resource consumption, of the function approximator. They suggest that there are few problems in which some knowledge is not available to the designer, and that this knowledge might be used to skew the function approximator’s resource allocation across the state-action space in order to achieve different degrees of resolution. They demonstrate good results using CMAC and other methods coupled with reinforcement learning in some simulated problems. However, as they mention, the skewing function they used was chosen by hand for each problem; should this skewing function be incorrect the resolution of the state-action space will not be correctly represented, impeding or even prohibiting learning.

Thrun and Schwartz [Thrun1993] make the point that since a function approximator introduces some noise due to over-generalization, when combined with a recursive value estimation scheme this may preclude the learner from achieving optimality for certain values of the discount rate λ. Smart and Kaelbling [Smart2000] show one solution to this problem in their HEDGER learning algorithm. HEDGER uses *locally weighted regression* (LWR) in which training points close to the query point have more weight than those further away. The function relating weight to distance is typically a Gaussian. When a new environmental condition is experienced, a prediction of Q-values can be generated based upon similar past experience. They present results in simulation and in a corridor-following robot-based task, in which they first ‘boot-strap’ the learner by allowing it to passively observe the decisions made by a training program or human operator, thereby updating its value function. In a second stage of learning they allow the learner to continue learning while in control of action selection. They report rapid learning which approaches the best results achieved under human control. In [Smart2002a] they expand upon this algorithm. Noting from [Gordon1999] that ‘*a function approximator can be safely used to replace the tabular value function representation if it never extrapolates from its training data*’, they check that new observations are within the training data already seen, and that the predicted value is within some accurate limits. Should these checks fail, a prediction is returned based upon the locally weighted average (LWA) instead of LWR. LWA is a function approximator that fits Gordon’s criteria. Smart [Smart2002b] shows that the algorithm learns faster when optionally employing LWR than one that relies upon LWA alone.

3.2.3 Learning Classifier Systems for Robots.

Cliff and Ross discuss results [Cliff1994] using ZCS in a simulated environment with continuous space and discrete actions of fixed size and Stolzmann presented results using a simulated Khepera to explore latent learning with the Anticipatory Classifier System (ACS) [Stolzmann1999] in which the environment was rendered into discrete states before presentation to the classifier system, but comparatively few accounts have been published on the application of LCS to the control of physical robots in the real world.

The most widely cited example of robot-based LCS learning is that of Dorigo and Colombetti [Dorigo1998]. In this work they present a novel LCS, based upon Holland’s original formulation. They make two classes of enhancement. Firstly, they address some of the problems in their original implementation of Holland’s system. Secondly, they demonstrate a parallel implementation of this algorithm which increases performance and lends itself to hierarchical decomposition of the learner to aid learning.

In $LCS_{0,}$ their original implementation of Holland’s LCS, they noted the following problems. Rules oscillated in strength, there was difficulty in regulating the interaction between the reinforcement system and the GA-based rule discovery system, chains of rules were unstable, and the system was slow to converge. They addressed these in their ‘Improved Classifier System’ (ICS) in the following ways;

- Overgeneral rules may advocate an action which is good in one state, but bad in another. In LCS_0 this resulted in such rules oscillating in strength, which would be used too often when they predict the wrong action, and not often enough when they are right. To solve this problem, they introduced the ‘*mutespec’* operator. The system monitors the variance in rewards that a classifier receives. If a classifier is judged to be oscillating (its variance exceeds some user-defined percentage of the population average), the mutespec operator is invoked. Unlike normal mutation, this changes ‘don’t care’ # symbols in the ternary condition representation to the specific ‘1’ or ‘0’; i.e. there is a chance of point mutation to a more specific representation.
- In most LCS, the genetic algorithm is called with some pre-defined, fixed frequency. As Dorigo and Colombetti point out, this frequency must not only be problem-specific, but also must vary according to the progress of the learning system in ascertaining the correct classifier strengths. As such, it should be adaptively controlled. They chose to implement this control on a systemic basis – it is difficult to see how it could be self-evolving as each individual would selfishly try to increase its own frequency of reproduction. In ICS the genetic algorithm operates when the ‘energy’ of the system has reached a steady state. Therefore the GA does not operate

when the sum of all classifier strengths is either increasing or decreasing over a sample period, nor when there are many oscillating classifiers.

- In ICS to speed convergence and reduce computational overheads very weak rules are deleted from the population. The rationale for this is that such rules are unlikely to be chosen by fitness-proportionate selection mechanisms, and if chosen would very likely advocate a less useful action. The authors claim that this speeds convergence since the population shrinks, and there are fewer rules to iterate through in set-based operations, thereby allowing faster iteration of the learning algorithm. While this may be very evident in simulation it seems unlikely to make a great difference in experiments on real robots where the amount of time to, for example, check whether a low strength rule matches the environment, is wholly insignificant compared with the amount of time that the robot takes to physically move from one state to the next.

Having addressed these shortfalls in LCS_0, the authors then present a parallel implementation of the ICS called ALECSYS. They implement parallelism at two levels. At the lower level, the basic LCS is parallelised. At the higher level, the control system for the robotic tasks is itself composed of multiple LCS which co-operate in a hierarchical arrangement in which sub-problems are addressed by separate LCS.

At the lower level, the population within a classifier system is split into sub-populations which are each handled on a separate processor. Many processes can be carried out without reference to global properties or lists, such as the composition of match sets. However, some activities such as the choice of action to be taken are carried out by a single process.

More interesting parallelism is evident at the higher level. Here the authors follow the familiar ‘divide and conquer’ approach to problem complexity by implementing a number of separate learning modules, each with the task of learning a different behaviour. These can be trained in ‘modular’ fashion whereby they each have a separate reinforcement function, or ‘holistically’ where the same reinforcement scheme is applied to all modules. In order to choose an action from the possibly conflicting recommendations of these behaviours, one or more further ICS are used as ‘switches’ which decide which of two modules should have its recommended action passed onwards towards the effectors, the other being suppressed. In this way, the architecture produced by Dorigo and Colombetti closely resembles that of Connell’s colony-style subsumption architecture [Connell1990] as used in Mahadevan and Connell’s paper above.

Dorigo and Colombetti apply this parallelised, hierarchical system to a number of real robot control problems of varying complexity. They use two training policies, either to reward the result (i.e. give reward when some goal is achieved), or to reward ‘the intention’. In the latter, small rewards are given when an action is taken that is consistent with a predefined model of the ‘correct’ state-action map. They tested both reward schemes in the generation of a simple light-following behaviour, and found that on this task performance was better with the reward the result policy.

In experiments with their ‘AutonoMouse’ robot, the environment was presented in discrete binary format, for example, a bit can be set to show that a light is in front of one of the eyes. Four discrete movement commands can be issued to each of the two motors; step backward, stay still, step forward, two steps forward. There are thus 16 composite actions which can be suggested by a behaviour module.

While the work presented in [Dorigo1998] is impressive, a number of points should be made;

- There is considerable reliance on *a priori* knowledge of the problem. This informs the hierarchical decomposition into behavioural modules, and the training policy used to reward the intention.
- Discrete inputs and actions are used. In effect, the robot is operating in a noisy grid-world. Once again, *a priori* knowledge must be used to establish the size of a 'square' in the 'grid' in which the robot is operating – how big is a step forward?
- The use of a 'reward the intention' policy which aids learning by providing continual reinforcement requires the experimenter to provide a training program that can assess the quality of each action taken by the learner. For some problems this could be easy to specify; the authors give the example of a 'follow the light' behaviour, where partial rewards are based upon a measure of the light intensity. They state;

  '*… that the distance between the robot's sensor and the goal should decrease is simply a formal statement of the specification of the light-following behaviour.'* [ibid., p. 177]

  However, this relies on much more knowledge than a scheme based upon delayed reinforcements. As the task becomes more complex, the reinforcement program will become more difficult to write, prone to error and therefore could result in the learning of behaviours which, rather than solving the overall task, profit from exploiting unforeseen local minima. Indeed they note that in some comparisons between 'reward the intention' and 'reward the result', performance was reduced with the partial reward strategy, stating that;

  '*… a trainer, in order to be a good trainer, needs very accurate low-level knowledge of the input-output mapping … in order to give the correct reinforcements.*' [ibid., p. 104]

The reader is referred to [Dorigo1998] for more details of this work.

Bonarini et al. detail a novel system called ELF, a controller based upon the evolution of fuzzy rules [Bonarini1994, Bonarini1996]. They note that in cases of delayed reinforcement, rules must co-operate together to solve a problem, but this is at odds with the fact that rules are competing for reward, may be over-general, and commonly in Fuzzy Logic Controllers (FLC) contribute to taking actions which are the combination of the recommendations of many rules matching at the same time (in fuzzy logic terminology these rules are said to have the same *antecedent* but different *consequents*).

Since rules that are triggered at the same time have the same antecedent, they compete with each other. Rules with different antecedents are triggered at different times, and therefore may co-operate together. Rather than evolving many complete FLCs – an approach they liken to Pittsburgh classifier systems, and which they ignore as unfeasible for evolving real robots – or evolving individual rules in a GA as used in Michigan classifier systems, they instead partition rules with the same antecedents together considering them to belong to the same sub-population, and run their evolutionary algorithm on each of these subsets in isolation. This is similar to the niche GA [Booker1985] as used in XCS [Wilson1995]. If there exists too many rules in an ELF sub-population, the worst are deleted, providing a fitness-proportionate selective pressure.

ELF uses a cover operator, and reinforcement with discounting in a similar fashion to Q-learning. There is dynamic resizing of the population according to measures of whether there are too many or too few rules. The GA uses only mutation of the consequents (actions) as an operator, and a rule is only considered for mutation if it has sufficient experience and has low fitness.

Reinforcement is carried out at the end of an ‘episode’ – a number of control cycles. The number of control cycles in an episode can be predefined, or can be triggered upon reaching a certain state. During an episode, only one rule in each sub-population can fire, and this rule is chosen randomly. Reinforcement is proportional ‘*…to a rule’s contribution to the obtained result*’. It also reinforces with discounting the rules that triggered in past episodes.

ELF continues in this way for a number of episodes. When the performance is satisfactory and the rule base has been steady for some period, the rule-base is stored and a random mutation is forced.

The ELF algorithm is an interesting approach that attempts to solve a number of specific challenges. It uses real-valued input, and in the animat experiments described in [Bonarini1996] using the ‘FAMOUSE’ agent, discrete actions (the consequents are limited to seven angles evenly distributed in the range $–180^o$ to $+180^o$ for steering). The authors describe good results in a task to follow a light. Later extensions to the algorithm included S-ELF [Bonarini1997], intended to co-ordinate predefined basic behaviours, and were tested on a physical robot called ‘CAT’.

The following points should be noted in relation to ELF;

- With random action selection, this may not be suitable for online control, though the final rule base may indeed produce excellent results. There can be few tasks we would want a robot to undertake where it could safely learn in such a way.
- This algorithm is not suited to lifetime learning. Each time a rule base is saved, there are random perturbations and performance drops. It seems well suited to training, followed by the cessation of learning. In contrast, and as outlined in the Introduction, the work in this thesis is motivated by the desire that a autonomous learner should not

> distinguish between a ‘training’ phase in which learning occurs and a ‘performance phase’ where no further adaptation is possible. While such a distinction may allow for the automatic discovery of solutions, it may become rigid and fragile when training ends since the system cannot adapt to changes in its environment.

Katagami and Yamada report work in which they attempt to speed learning by bootstrapping an LCS through interactive human training [Katagami2000, Katagami2001]. The environment is presented to the classifier system as 16 binary characters where the first 8 correspond to the robot’s eight IR proximity sensors, and the second eight represent the robot’s eight light detectors, these being set to 1 if some predetermined threshold value is exceeded. The environment as perceived by the robot is presented to the operator who can guide the actions of the robot using a joystick. If the state-action pair thus generated is not represented by a classifier in the rule-base, a new matching classifier is created. If there is a classifier which matches, its numerosity is increased by one. Their LCS is based upon Wilson’s XCS [Wilson1995]. The learner performs alternating taught and autonomous trials. They present results which show an improvement in the speed of learning with teaching, compared with the LCS alone.

One problem with this approach may be that the environment as presented to the classifier system is represented as, for example, an array of real numbers, which may not be readily comprehensible by a human operator. In Katagami and Yamada’s work, they allow the human operator to see both video from the robot’s perspective and the sensory data which is fed to the classifier system. A human trainer using rich video data may be able to discern environmental cues which are not available to the classifier system which is presented with much simpler input; this problem will not be apparent in simulation since a visual representation of a grid world contains no more data than the description of the environment which is presented to the classifier system. However, they show better performance if the robot is trained with an operator using the easily-understood video, rather than the robot's sensory data.

We have seen that evolutionary methods generally take many hours to produce robotic controllers, unless partially or totally trained in simulation (or trained in real-time by a human operator). We have seen that reinforcement learning techniques commonly require the discretisation of the input space, which may compound the problem of perceptual aliasing.

Ideally, a mechanism for producing robot controllers should show the following desirable characteristics;

- Life-time learning to enable the controller to adapt to changes in its environment, rather than separate 'train' and 'perform' phases.
- Simulation should not be necessary.
- The system should be able to deal with delayed reinforcement. Complex problems that must be solved by the composition of behaviours may not be easily represented by a 'reward the intention' system of continual assessment.
- It should not take excessive time to produce optimal, or near optimal, behaviour.
- It should not be necessary for the experimenter to decide *a priori* schemes for reducing the environment's complexity, since such schemes may be inaccurate, hide vital environmental cues, and may stop the controller from adapting to changes in its environment. Therefore neither the input or output space should be manually discretised.
- The system should be robust in the presence of noise, both externally and internally generated.

3.2.4 Introduction

Hurst et al. presented work in which they applied two classifier systems to problems of robotic control [Hurst2002a, Hurst2002b, Hurst2003]. They first demonstrate that ZCS can be used successfully to learn an obstacle avoidance behaviour on a LinuxBot (see section 3.4.1). The robot has three IR collision detectors, one facing forward, one facing slightly to the left, and to the right. These can be on or off, providing a simple three-character representation of the environment. Actions are encoded as a single integer – this represents the continuous actions turn left, go forward, or turn right. The classifier system chooses an action in response to environmental input, and the robot initiates this action. The ZCS algorithm then pauses until a change in state is observed; i.e. the environmental input has changed, and a new cycle starts with the production of the match set. This is similar to the application of ZCS in a simulated continuous environment by Cliff and Ross [Cliff1994]. A reward of 1000 is given when the robot is in clear space, and to encourage the robot to continue in clear space for as long as possible, '*if the last reward was the maximal reward the reward given to the robot is increased by 100*' [Hurst2003]. This occurs when the robot has continued to move in free space for more than some time-out value. A diagrammatic representation of ZCS as used here is presented in Figure 3-1.

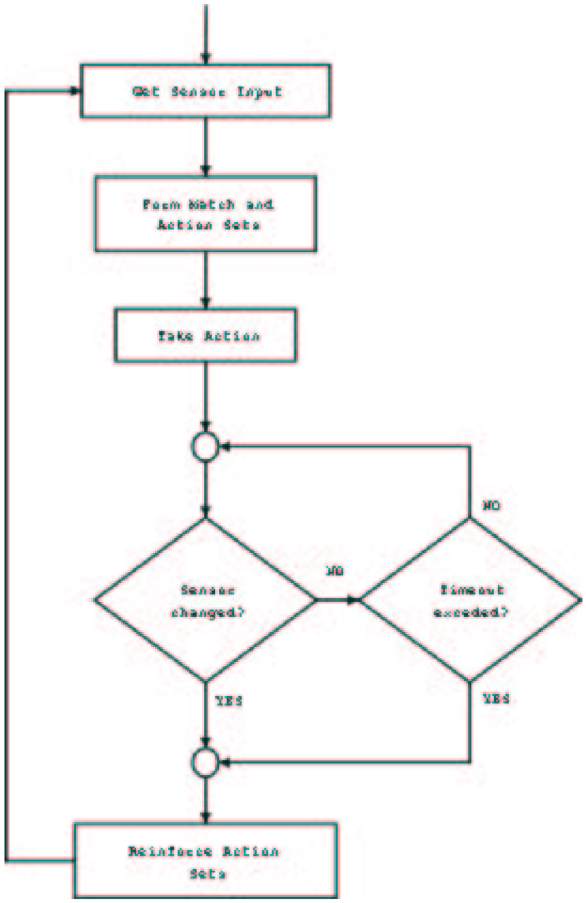


Figure 3-1 'Event'-based extension of ZCS, after Hurst et al, 2003

Concentrating on the concept of 'events' allows ZCS to use continuous actions, rather than actions which move the robot a predetermined distance. Once an action has been selected it is performed until environmental change is perceived, or the time-out value is reached. As the sensory input is composed of discrete binary values, it is easy to determine when an event has occurred.

However, consider the case where the environment is represented by continuous values. It would be possible to impose a discretisation upon it, rendering the problem similar to the familiar grid worlds. As discussed above, this is undesirable; incorrect *a priori* assumptions may make optimality unachievable. Hurst et al. addressed this problem with TCS – a Temporal Classifier System – which incorporates an approach that tackles Semi-Markov

Decision Problems within the Reinforcement Learning framework. This addresses a second problem; how to choose between actions that move the learner from one state to the next, but take different amounts of time to complete.

3.2.5 Reinforcement learning in a Semi-Markov Decision Problem

Consider the case where more than one action can move the robot from one state to another. In the algorithm as presented above, reinforcement occurs when a new environmental state has been achieved, or when the desired state of having the bumpers clear has been maintained for more than the timeout value. If the robot could move from having a bumper triggered to having no bumpers on in two ways, one only taking seconds, but the other taking hours, both actions would receive the same reinforcement and would thus appear equally good.

In such circumstances, where in effect a continuous environment must be rendered into discrete states in order to use a reinforcement learning algorithm, the value of a state-action pair should be related to temporal considerations. This sort of problem is known as a Semi-Markov Decision Problem; an excellent overview can be found in the work of Sutton et al. [Sutton1999]. As stated by Parr [Parr1998], *'…an SMDP is just like an ordinary MDP, with the difference that transitions may have a stochastic time duration'*.

As noted in section Chapter Two, the reinforcement mechanism implemented in ZCS closely resembles the *Sarsa* algorithm, a development of the *TD(0)* algorithm which derives the value of state-action pairs in place of states. In *Sarsa*, the update equation is as follows;

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \qquad (1)$$

The external reward *r* is discounted by *α* and the reward from the next state, if any, is discounted by *γ*α*. Neither of these rates is influenced by time. Parr [Parr1998] presents a modification of the *Q-learning* algorithm in which both the external reward and the operation of the 'bucket brigade' are effected by time.

'On a transition from state s to s' under action a that has taken time t and received reward r (which is assumed to be the appropriately weighted sum of rewards received during t):'

$$Q(s,a)^i \longleftarrow Q^{i+1}(s,a) + \alpha^i(s,a)\left(r + \beta^t V^{i-1}(s') - Q^{i-1}(s,a)\right) \qquad (2)$$

where β is a discount rate that varies both with state and action.

Hurst et al. incorporated these changes to the bucket-brigade into ZCS in the following ways.

Firstly, the external reward is discounted according to the amount of time taken to reach the goal, t^t. This will favour efficient over-all solutions.

$$r = e^{-\sigma t^t} r \qquad (3)$$

Secondly, the discount factor *γ* also factors in time, but in this case the time taken to make the transition *between* events, t^i. This will favour actions that transition between states efficiently.

$$\gamma = e^{-\eta t^i} \qquad (4)$$

In effect, *σ* and *η* are different learning rates that change the emphasis placed upon the overall time to achieve external reinforcement, and the time taken performing individual actions.

The original ZCS update algorithm [Wilson1994] was;

$$S_{[A]} \xleftarrow{\beta} r_{imm} + \gamma S_{[A]'} \qquad (5)$$

in which $S_{[A]}$ is the current action set, r_{imm} is the immediate reward, $S_{[A]'}$ is the next action set, and $\xleftarrow{\beta}$ is the Widrow Hoff gradient descent procedure with the learning rate β. Incorporating (3) and (4) into (5) gives the update procedure that Hurst et al. used in TCS;

$$S_{[A]} \xleftarrow{\beta} e^{-\sigma t^{t}} r_{imm} + e^{-\eta t^{i}} S_{[A]'} \qquad (6)$$

In summary then, there are now two forms of discounting; external reward is dependent on the total time taken to achieve the goal, and the amount of reinforcement flowing to previous action sets is proportional to time, favouring longer actions over short ones.

3.2.6 Algorithmic description

By incorporating Parr's equations relating to SMDPs into the ZCS reinforcement algorithm, Hurst et al. addressed the second of the problems outlined at the close of section 3.2.4; real time is now considered in the reinforcement scheme. They also addressed the first problem, namely, how to define events in a continuous space with real-valued inputs.

It will be recalled that in applying ZCS to the problem of robotic control, the binary nature of the sensor array rendered the environment as an aliased discrete space. Events were simply determined as changes in this discrete representation; this discretisation being imposed by the hardware that was used to generate the environmental representation. This led to a small state space, which promotes ease of learning as even without the generalizing abilities of a classifier system, the state-action space is small (in this case, 2^3 input bits describe state, and there are three possible actions, so the Q-table comprises 24 values).

With an environment that is represented with real numbers, the problem becomes much trickier. We have seen that one common approach to a complex environment is to impose a hand-crafted discretisation upon the input, declaring in effect that the continuous world can be reduced in complexity to ‘blocks’ of a predetermined and fixed size. The onus is upon the experimenter to decide what resolution of discretisation is appropriate for a problem, but the abstraction suited to one environment may not suit another, and robot hardware may differ significantly, one platform from another. This approach is thus inflexible, and requires the experimenter to attempt to ‘view the world through a robot’s eyes’; the values returned by physical sensors may not be as expected.

Alternate approaches within the reinforcement learning literature such as that described by Uchibe et al. [Uchibe1997] rely upon pre-processing the complex environmental representation to build a model, and then applying reinforcement learning to this model. Sutton and Barto suggest two approaches to the problem of non-Markov environments [Sutton]. The first is to use pre-processing of the environmental input to render it Markov; for example, Bayesian methods can be used to compute at each time step the probability that the environment is reducible to an underlying Markov Decision Problem. The second approach is to use some sort of aggregation technique, for example, a function approximator such as a neural network, to treat a set of observable states as a single state to present to the reinforcement learning system. They make the point;

‘… the overall problem divides into two parts: constructing an improved representation, and making do with the current representation. In both cases, the ‘making do’ part is relatively well understood, whereas the constructive part is unclear and wide open.’

This then is the context in which TCS was developed. In order to define automatically what constitutes an ‘event’, the action-selection sub-system of ZCS was extended. Initially, a match set is formed in the same way as in ZCS, and thence an action set. The action is taken. In a new ‘drop decision’ cycle, the environmental representation is repeatedly presented to the members of the action set [A]. If a time-out value has been reached, or an external reward has been received, the drop decision cycle terminates. In the case that the timeout value has been reached, the members of [A] are not assigned to the previous action set [A_{-1}] so they receive no update in the next cycle. If an external reward is achieved, the members of [A] are rewarded, and the next cycle of sense, match set [M] formation, etc. starts.

If neither external reward nor time-out has terminated the algorithmic cycle then a decision is made on whether the classifiers in [A] continue to match the current environment. Two sets are formed, the ‘drop set’ of classifiers in [A] which no longer match, and the ‘continue set’ of classifiers which do. The drop decision cycle is illustrated in Figure 3-2, and occurs as follows;

- If no classifier matches the current input, i.e. [continue] is empty, then the drop decision cycle ends, and the familiar stages of reinforcement (and optionally the GA) are performed, [A_{-1}] being set to [A].
- If all classifiers match the current input, i.e. [drop] is the null set, the cycle continues, and thus the original action carries on.
- If some classifiers are in [drop], and some are in [continue], a decision must be made on whether to continue or terminate the current action. This decision is made on the strengths of the classifiers, and may be done stochastically using e.g. roulette-wheel selection on [A] to pick a classifier, or deterministically by picking the strongest. If the classifier thus chosen is in [continue], [A] is set equal to [continue]. The classifiers in [drop] will not receive an update via the bucket brigade. If the picked

classifier is in [drop], [A] is set equal to [drop], the classifiers in [continue] will be excluded from the bucket brigade updates, the current action is terminated and the processes of reinforcement etc. take place.

It should be noted that TCS achieves operation in continuous time and space by two different methods. It operates in continuous time as reflected through the two discounting methods, but deals with continuous space by the automated disovery of useful interval representations; the discovery and refinement of hypercubes.

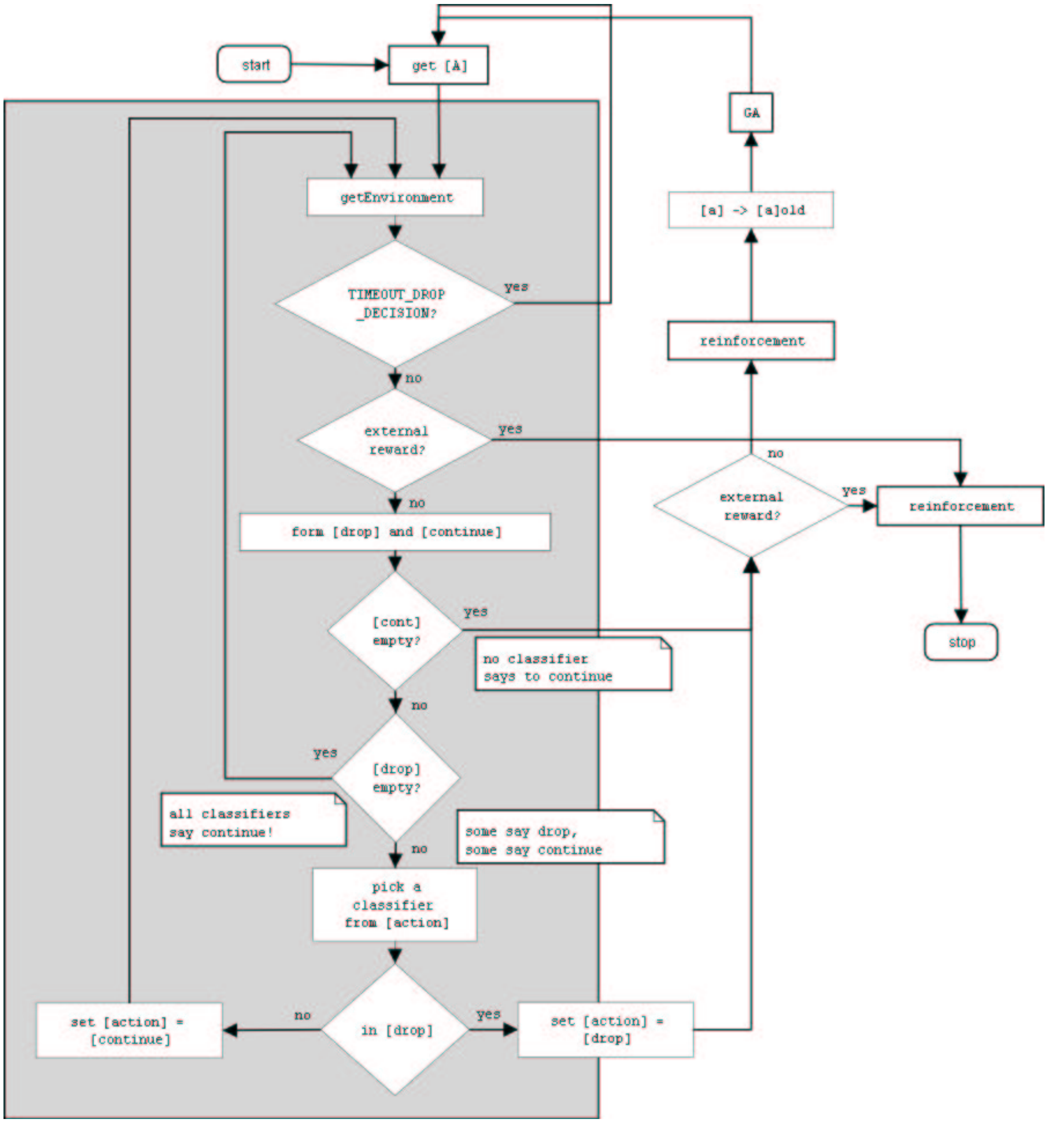


Figure 3-2 The TCS action-selection mechanism. The shaded area represents the 'drop decision' cycle.

3.3 Experimental set-up.

3.3.1 Hardware : The LinuxBot.

The hardware platform used for the experiments reported in this chapter is the same type as that used by Hurst et al., that is, a ‘LinuxBot’. The LinuxBot [Winfield2000, 2003] consists of a wheeled platform with a driven wheel on each side and a trailing tail wheel. On this platform sits a miniaturised PC running a distribution of the Linux operating system, in this case Slackware 7.1. The PC is networked using a Wireless Ethernet card that allows remote operation via Telnet, etc. A combination of battery and electrical pickups that contact the floor surface provide power for hours of continuous autonomous operation.

The LinuxBot may be equipped with a variety of different sensors, including IR receivers, IR proximity detectors, physical bumpers operating switches, and light sensors. In the experiments presented here, the robot was equipped with three IR proximity detectors, or ‘IR bumpers’, one facing forward, one forward and slightly to the left, and the third slightly to the right. Two physical bumpers are mounted at the front of the robot, one wrapping around to the left side, and one to the right. The robot is equipped with an elevated mast on which three light-dependent resistors (LDRs) are mounted. Again, one faces forward, one is angled 45^o to the left, and one 45^o to the right. This arrangement is shown in Figure 3-3, in which the IR bumpers are pink, the LDRs are blue, and the physical bumpers are red. The LDRs and various bumpers can be clearly seen in Figure 3-4.

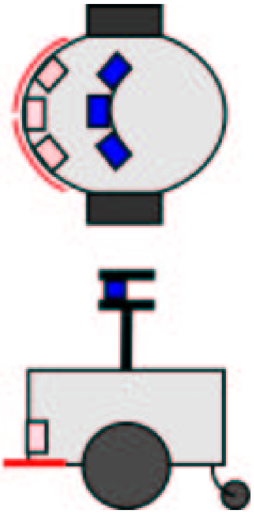


Figure 3-3 The LinuxBot (plan and side elevations) showing bumpers and light sensors (see text).

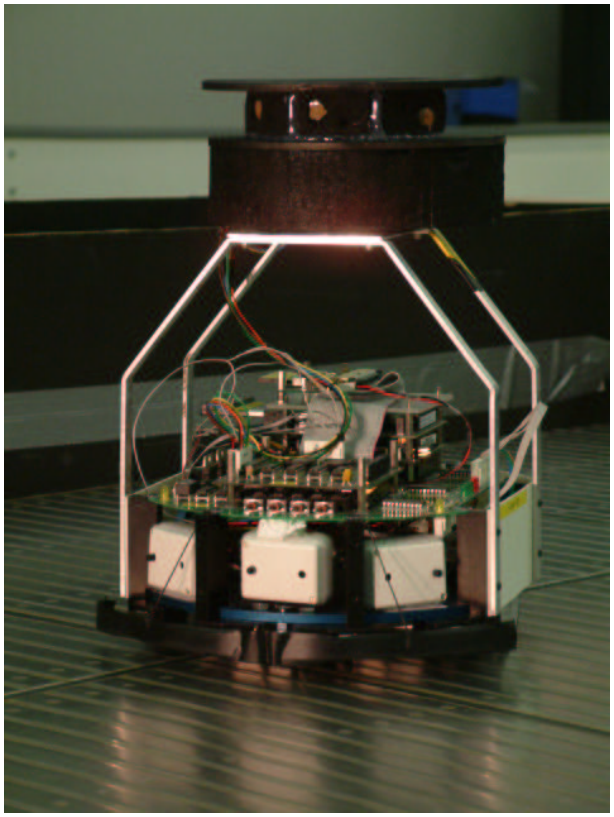


Figure 3-4 The LinuxBot on powered floor.

The LinuxBot is approximately 30 cm in diameter, and 40 cm in height from the floor to the top of the LDR stack.

3.3.2 Environment.

The environment in which the LinuxBot performed the learning experiments to be described in Section 3.4 consists of a square enclosure, measuring approximately 270 cm on each side. The walls of the enclosure are approximately 25 cm in height, and are painted matte black. The floor of the enclosure is composed of tiles that have alternating positive and negative DC current; this power is intermittently available as the robot moves around the arena, and is used to extend the operating time by recharging the battery via the downwards-pointing electrical pickups.

There is a light source (halogen lamps) at one end of the enclosure, approximately half way between the two sides.

The tiles of the powered floor are supplied with electricity using computer-controllable power supplies. All tiles are powered, but three of the tiles are powered by their own power supply, which supplies no other tile. By monitoring the current drain on these three power supplies using an RS232 serial communication link, a PC can signal to the LinuxBot when it is on one of these tiles via the wireless LAN. This is illustrated in Figure 3-5, in which the lamps are shown as yellow circles, and the three monitored power supplies are shown as black, white and grey, supplying their corresponding square.

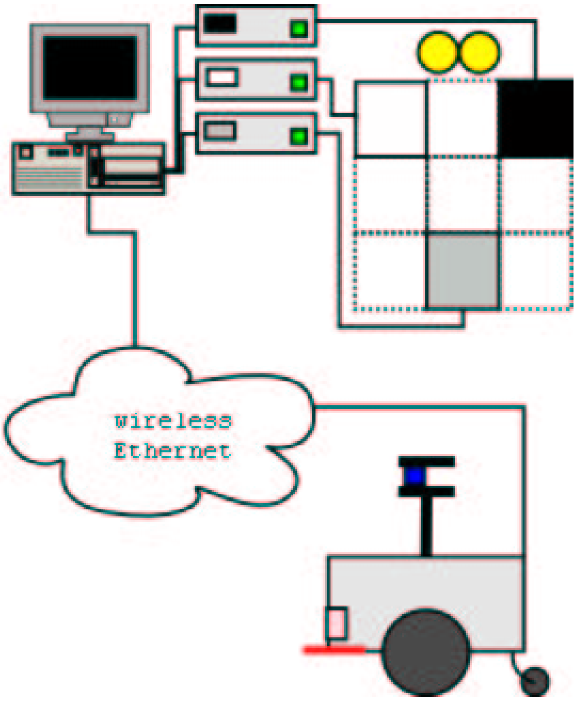


Figure 3-5 The arena, showing lights and power.

The black and white tiles to either side of the lamps provide two possible goal states. The grey tile is ‘home’, i.e. the tile to which the robot must return before starting a new learning trial.

Due to some peculiar inadequacy of the communications system on the power supplies, the monitoring PC is unable to poll them too quickly. Due to this, there may be a delay of up to one second between successive readings being reported from a power supply. This introduces noise into the reward signal.

No special measures were taken to insulate the environment. The arena is open to natural diffuse light, and is situated next to a busy corridor with passing staff and students. The laboratory in which the experiments took place is large, and there may be other experiments involving halogen lamps switching on and off outside our control. Any learning must therefore cope with quite a high potential of noise, in addition to the noise introduced by the LDR sensors themselves.

Although the environment has tiles which represent reward states, it should not be considered to be a ‘real grid world’. There is no grid-like discretisation of the input to the system, the actions perfromed by the robot are not of fixed size, and as stated above, the reward signal is noisy.

3.3.3 Software.

TCS is presented with an environmental input consisting of three real numbers representing the resistance of the LDR sensors, scaled between 0.2 and 0.8. The condition part of the classifiers is encoded as un-ordered pairs of real numbers in the range [0, 1], one pair for each environmental input. A pair is considered to match the corresponding input value if one of the pair is smaller or equal to the target, and the other is larger or equal. The action of the classifier is an integer, representing the actions move forward, turn continuously to the left, and to the right.

An unordered representation was chosen for the condition pairs based upon the work reported by Stone and Bull [Stone2003]. In extending XCS for use in real valued environments, Wilson has suggested two representations; ‘Centre-spread’ [Wilson2000] and ‘Lower-Upper Bound’ [Wilson2001]. In the ‘centre spread representation’ the condition has a pair of numbers for each environmental input. The two numbers encode respectively a real number (the ‘centre’), and the ‘spread’ on either side of this number within which the corresponding environmental input is said to match, i.e. the environmental reading e is said to match the tuple (c_1, c_2) if $c_1 - c_2 <= e <= c_1 + c_2$ In the ‘Lower-Upper Bound’ representation, the numbers in the condition pair (c_1, c_2) match a single environmental input e if $c_1<=e<= c_{2.}$ Stone and Bull show that the ‘Centre-spread’ representation introduces significant bias, and point out that the 'Lower-Upper Bound' representation demands that any operation that could result in $c_1 > c_2$ must result in a reordering of the alleles. To simplify this, they introduced the 'Unordered Bound Representation' as used here, which they show has the same desirable properties as the 'Lower-Upper Bound' scheme.

One problem with the 'Unordered Bound Representation' is that values at the extremes of the solution space are much less likely to be covered. Consider the case where input is scaled between 0 and 1, as are the numbers of the condition representation. An environmental input of 1.0 can only be matched by a classifier having a 1 as one of the two unordered pair. Values in the centre of the range are more likely to be matched. To address this, the environmental input was scaled in the range [0.2, 0.8].

The crossover and mutation operators are altered from those in ZCS to deal with the new condition. In crossover, there is an equal chance of crossing over at any point in the condition. Mutation can occur at any point in the classifier with probability μ. The real numbers of a classifier's condition are mutated by a fixed small change of ± 0.005. Action mutation is by picking an integer from the set {0,1,2} at random, such that the chosen action is different from the current one.

In the initial population, classifier conditions are created randomly in the range [0,1]. During cover, the current environmental input e is used as a centre and two values are created in the range $[e\text{-}C_{max}, e+ C_{max}]$, where C_{max} is 0.2

To speed convergence, the classifier from [A] which determines whether the drop decision terminates is selected deterministically on the basis of fitness, rather than by using roulette-wheel selection.

In addition to the conditions of the drop decision described by Hurst et al. and detailed here in Section 3.2.6, the drop-decision cycle is also terminated if the bumpers are on. In this case the *last* drop-set is returned as the set which will receive reinforcement to discourage the robot from walking into obstacles. If there is no ‘last drop-set’, the null set is returned.

3.3.4 A ‘Trial’

In the experiments described here, the system is first calibrated on the light readings immediately next to the light source, and pointing in the opposite direction. To reduce the impact of noise from the LDR sensors, these readings are the average of 100. The robot is then placed on the ‘home’ tile, and the first learning trial begins. TCS is stopped when the robot reaches an appropriate ‘goal’ tile. The learning algorithm is then stopped, and the robot is placed under the control of a ‘Braitenberg Vehicle-style’ algorithm [Braitenberg1984] that performs obstacle avoidance, ‘bouncing’ around the experimental pen until the robot is informed that it is on the ‘home’ tile. Once there, the robot orientates itself approximately towards the light, and the next trial begins. In order to save time, any trial that lasts for longer than 30 seconds is considered to have ‘timed out’ – there is no reinforcement, and the robot return to start a new trial as above.

3.4 Results

3.4.1 One Objective

The performance of the TCS algorithm is first demonstrated on the robotic platform in a single objective task. Only one of the ‘goal’ tiles causes an external reinforcement to be applied, thereby ending a trial. The external reward is 1000.

The robot may start at any point on the ‘home tile’, and so the distance to the goal may vary between trials by approximately 80 cm. Also, as mentioned above, delays in polling the power supplies may mean that the robot is on the goal tile for up to a second before this fact is signalled to TCS. These two factors mean that the ‘optimal’ time to complete a trial should be considered as a range of values; in the best case, approximately six seconds, and at the other extreme about nine seconds.

The following parameter settings were used on this single objective task.

N	600
S_0	10
β	0.2
σ	0.1
η	0.7
τ	0.1
χ	0.5
μ	0.05

Table 3-1 Parameter settings for single objective TCS task

TCS was found to easily achieve good results on this single objective task, comparable to those reported in [Hurst2003]. In the Figure 3-6 and the accompanying Figure 3-7 the average results of five runs are presented, each run lasting for 200 trials. In Figure 3-6 it can be seen that TCS is soon capable of performing nearly optimally, finding the goal each time with only an occasional failure. Figure 3-7 shows the windowed average of the percentage success,[12] windowed over 50 trials, and the average time taken to reach the goal. After a short initial period in which performance fluctuates, the algorithm quickly shows that it is capable of nearly optimal performance. There is a strong trend for the time taken in each trial to reach the goal to be within the range considered optimal.

In order to complete a run of 200 trials, the experiment must run for approximately three and a quarter hours.

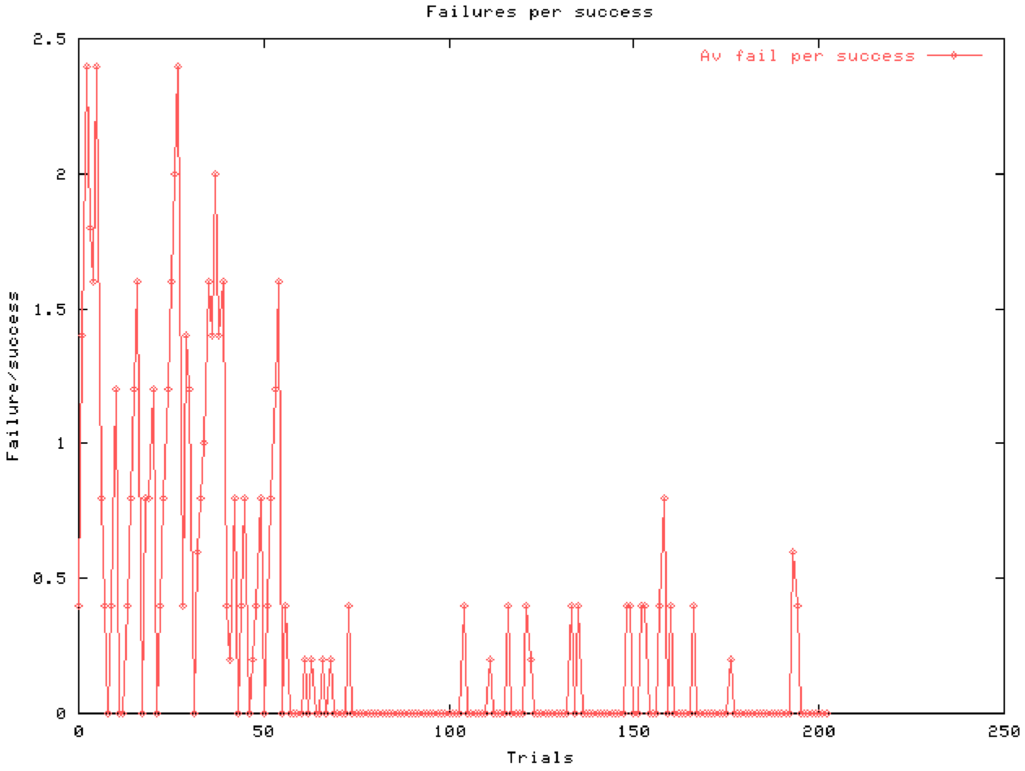


Figure 3-6 Failure to Success ratio

[12] The step in the graphs of percentage success is an artefact of the graphing package. Since a windowed average is used, there is no data for the first *n* trials, where *n* is the window size; this is shown as a line at 0, followed by a sudden jump when *n* is reached.

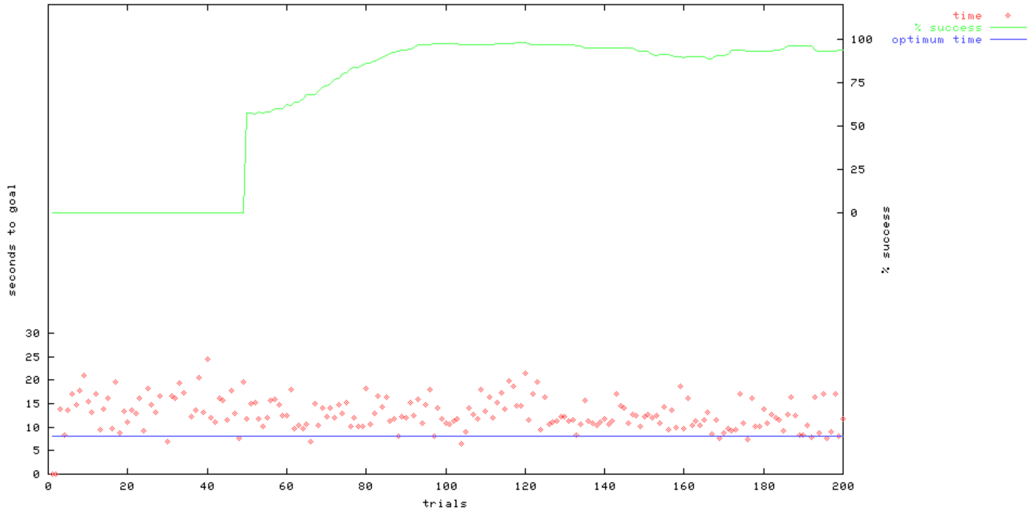


Figure 3-7 Percentage Success and time to goal

In Figure 3-8 and Figure 3-9 we see graphs of individual trials. Once again we see that TCS is capable of achieving the goal nearly all the time. It is also apparent that the time taken to reach the goal is falling towards the optimal band. Figure 3-9 shows a longer run which demonstrates that TCS is capable of maintaining successful behaviour.

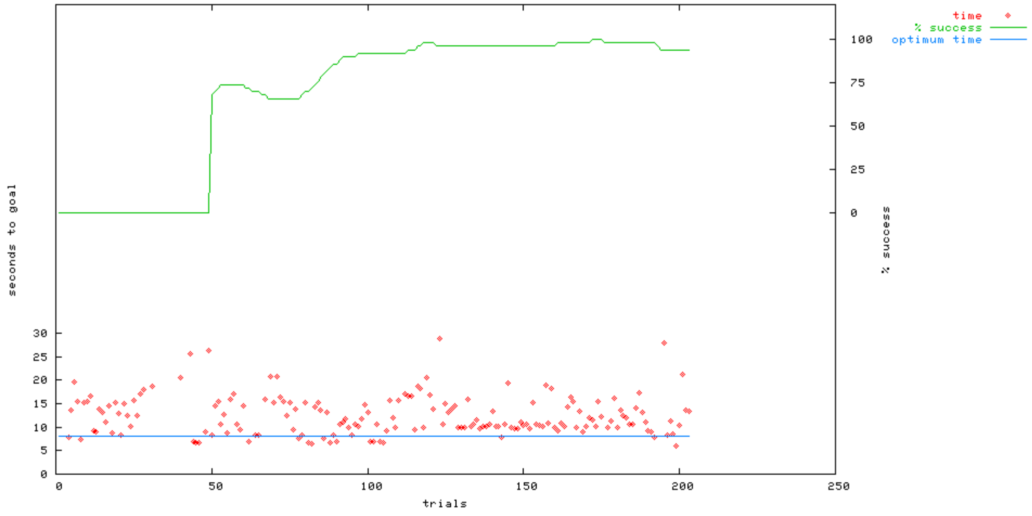


Figure 3-8 Percentage Success and time to goal.

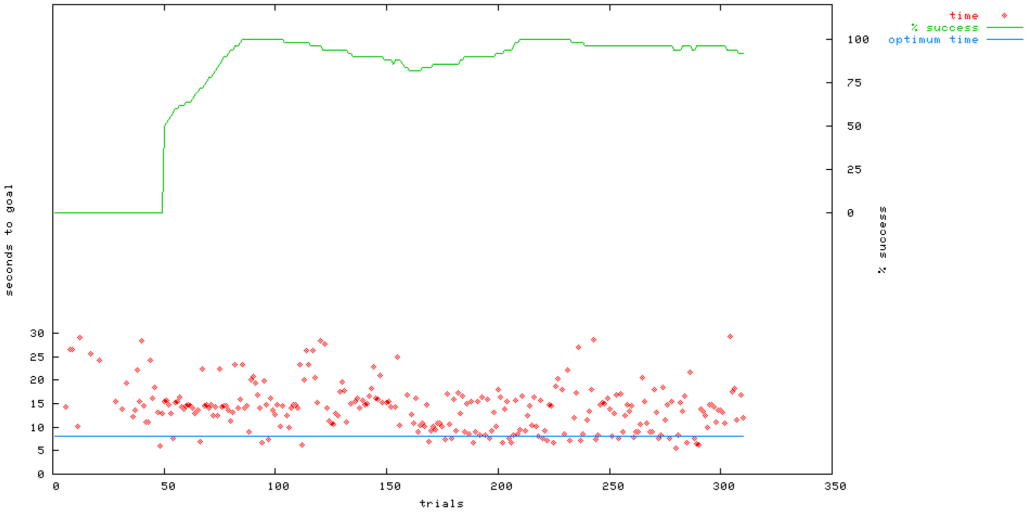


Figure 3-9 Percentage Success and time to goal.

It should be noted that the settings of the reinforcement parameters were based upon those reported by Hurst et al. in their ‘towards the light’ experiment. The system may have achieved better or worse results with different settings.

3.4.2 Two Objectives

The experiment was then expanded to include two goal states, which shall be referred to as ‘back’ and ‘white’ as shown in Figure 3-5. The classifier condition is extended by an additional unordered pair of real numbers, which are tested to see if they match the robot’s ‘internal energy level’. This level is set at the beginning of each trial so that 50% of the trials have an ‘energy level’ of more than 0.5, and the rest are lower than 0.5. The reward scheme resembles that in the simplest of the three concurrent dual-objective tasks presented in Chapter 2, i.e. a high reward (here 10000) is discounted according to the time taken to complete the trial and given when the robot has been guided to the correct goal (see section 2.7.1). A low reward (10) is similarly discounted according to the time taken to reach the incorrect goal, should the trial end there. No reward is given if the trial times out without either goal being reached. The energy level of the robot is static.

The results TCS achieved are not optimal. There is considerable variation between different runs. In Figure 3-10 we again see time to goal, broken down by the goal state achieved. These figures also show a windowed average of the percentage of trials that are successful. A trial is considered successful if the robot reached the goal which was appropriate for the ‘internal energy level’.

All parameter settings remain the same as in the single objective trial, except the population size was increased. If 600 classifiers were used to correctly encode the relationships of states and actions for a single objective task, it seemed realistic to expect that double this number would be required for the dual objective task (of course, generalization over the ‘energy level’ might allow this number to be reduced in the early stages of navigation when the robot is far from the goal states).

In Figure 3-10 we see that there is little sign that the time to goal is decreasing towards the optimum. The percentage of trials in which the robot achieved the correct trial increases slowly.

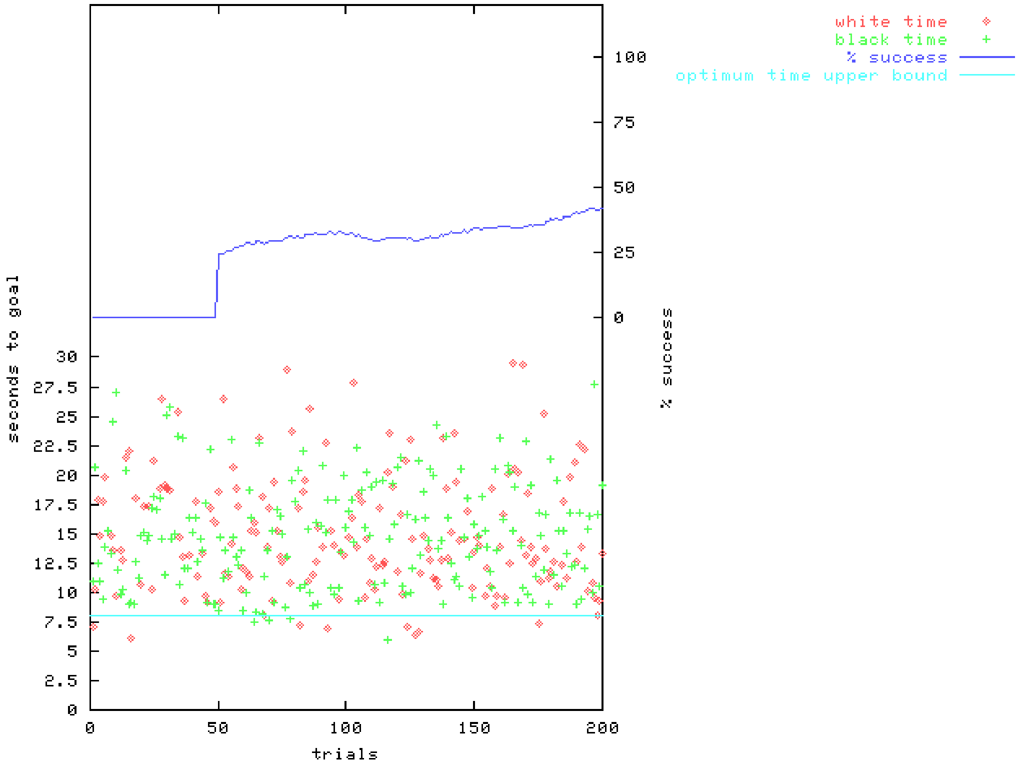


Figure 3-10 Percentage Success and Time to Goal.
Two objectives. Average of 10 trials

We have seen in the previous chapter that optimal behaviour with ZCS is very dependent on the settings of the reinforcement learning parameters, in addition to the population size. Figure 3-11 and Figure 3-12 present results from two runs in which the experiment was allowed to continue for longer, and in which the population size was increased to 2000. The values of the temporal discounting parameters were also changed, in order to try to force TCS towards optimality in the time taken to reach the goals, setting σ to 0.05 and η to 0.1.

Although in Figure 3-11 we see a slow increase in percentage success, there is little sign that optimal time to goal will be achieved. Figure 3-12 is even more disappointing, since on average the robot reaches the wrong goal more often than the right one!

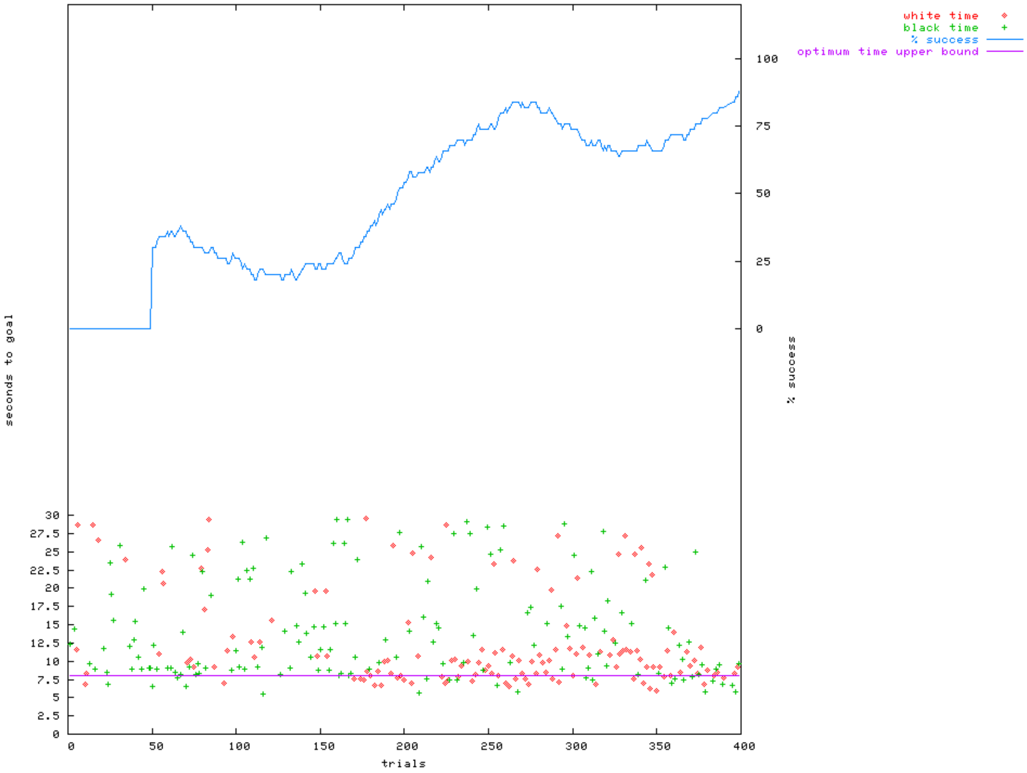


Figure 3-11 Percentage Success and Time to Goal. Two objectives.

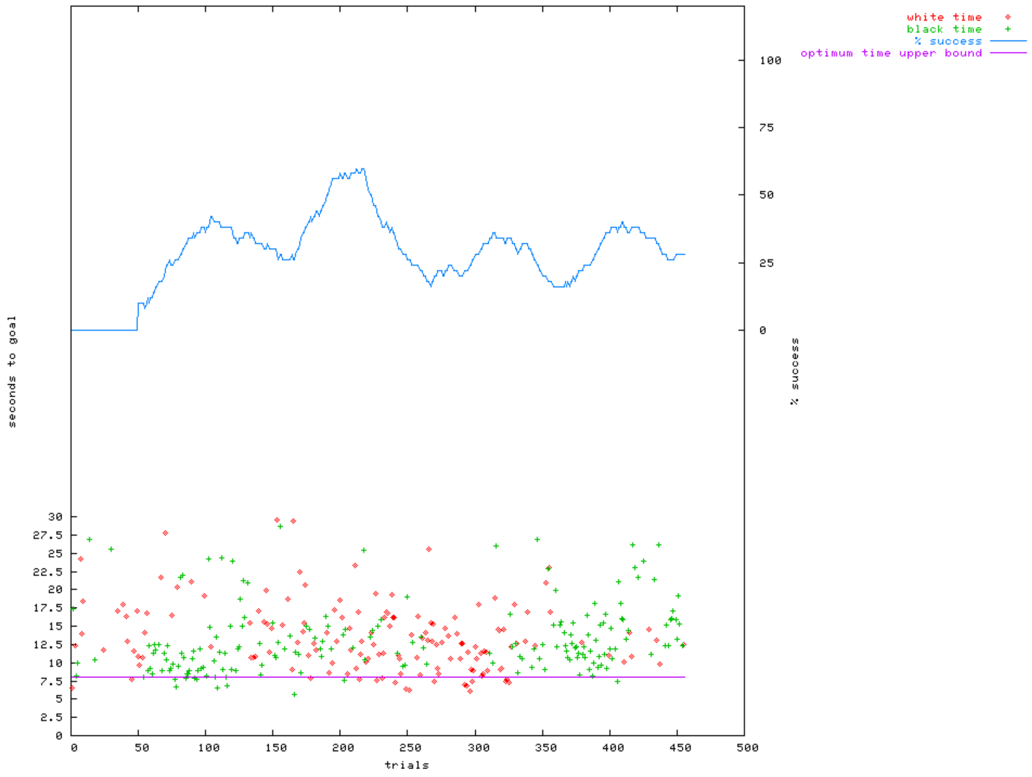


Figure 3-12 Percentage Success and Time to Goal. Two objectives.

Keeping the same parameter settings, the system was then run for 1000 trials. Due to restrictions beyond my control, it was not possible to run the robot continually overnight, and so these experiments had to be done in stages. The population was stored, the robot stopped, and then the system started again the next day. In order to complete 1000 trials in a single run, the robot had to operate for approximately 20 hours, spread over 4 days.

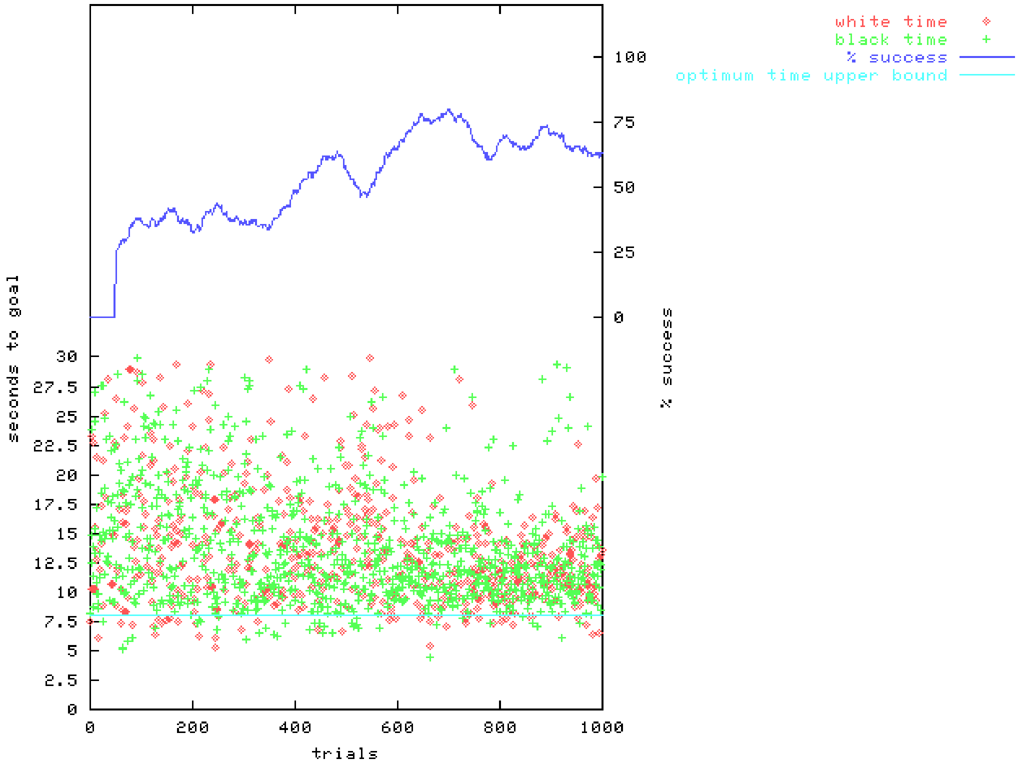


Figure 3-13 Dual objective, 1000 trials. Average of 5 runs.

Figure 3-13 shows the average of five runs, each run lasting for 1000 trials. With more trials, there is more evidence that TCS is able to choose the correct goal dependent on the internal energy level, and there is clearly a trend towards faster solutions. Even with a much longer experimental run, it was not possible to achieve near-optimal performance with TCS on this simple dual objective problem. The behaviour of the robot seemed consistently to 'concentrate' on one goal, and then switch to the other. This can be seen in the figures below. While some experiments (e.g., Figure 3-14) showed progress towards successfully reaching the correct goal each time, others were much less good, e.g., Figure 3-17. In no case can we see a convincing case for an approach to optimal time.

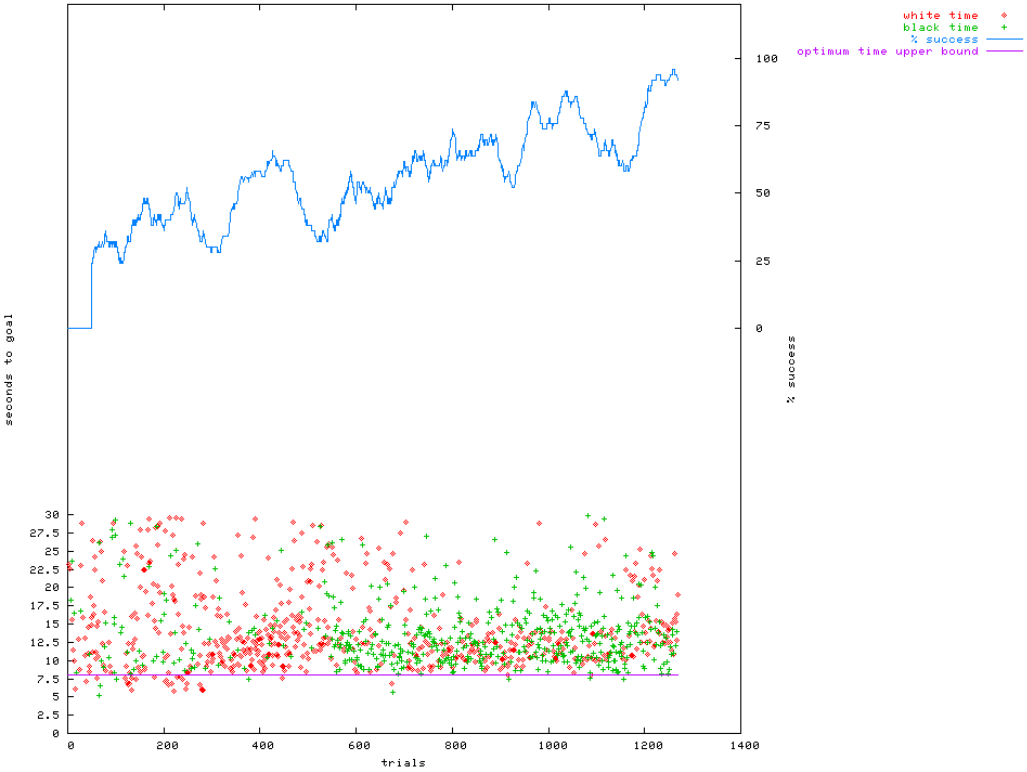


Figure 3-14 Longer run of TCS on the dual objective problem.

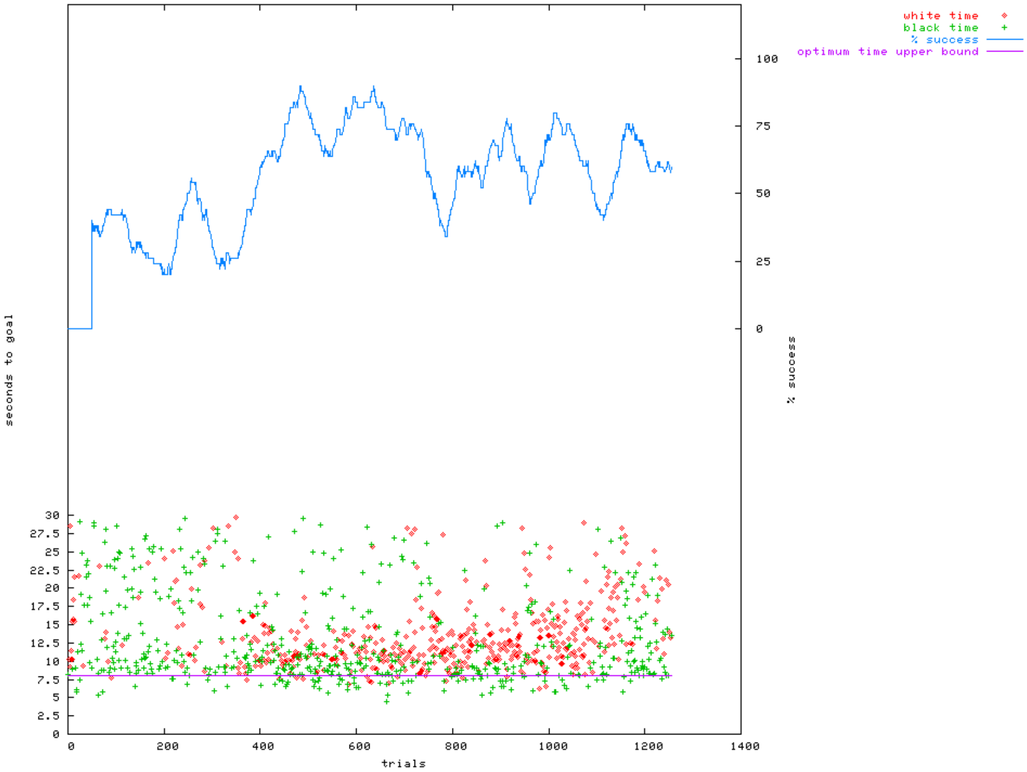


Figure 3-15 Longer run of TCS on the dual objective problem.

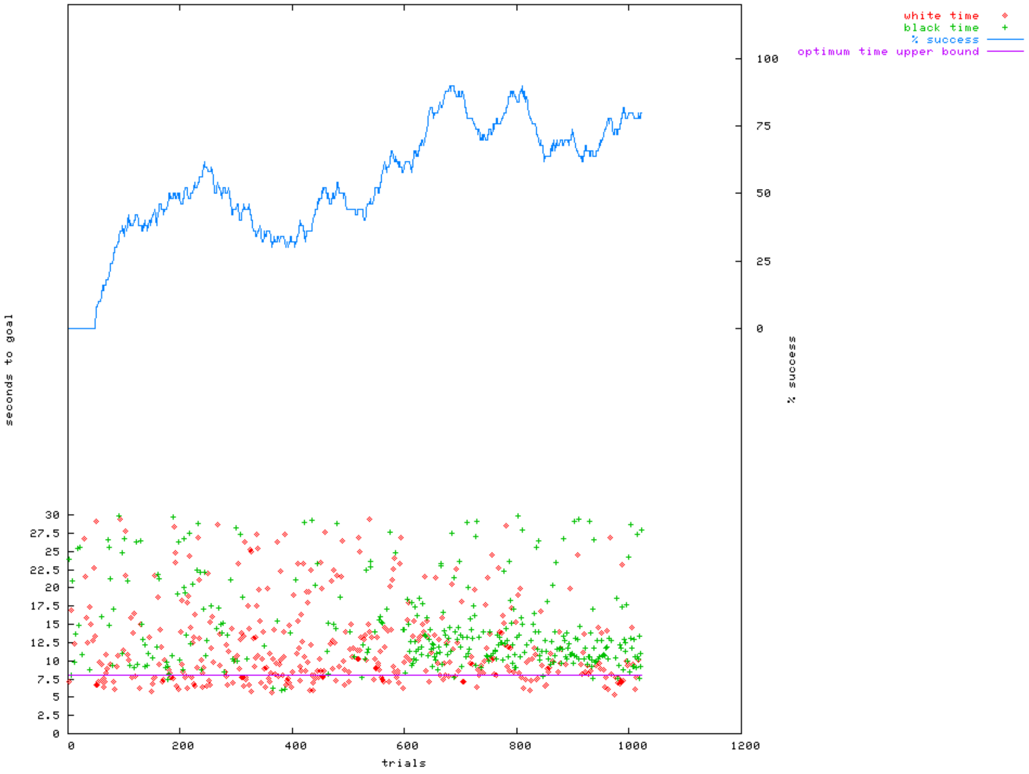


Figure 3-16 Longer run of TCS on the dual objective problem.

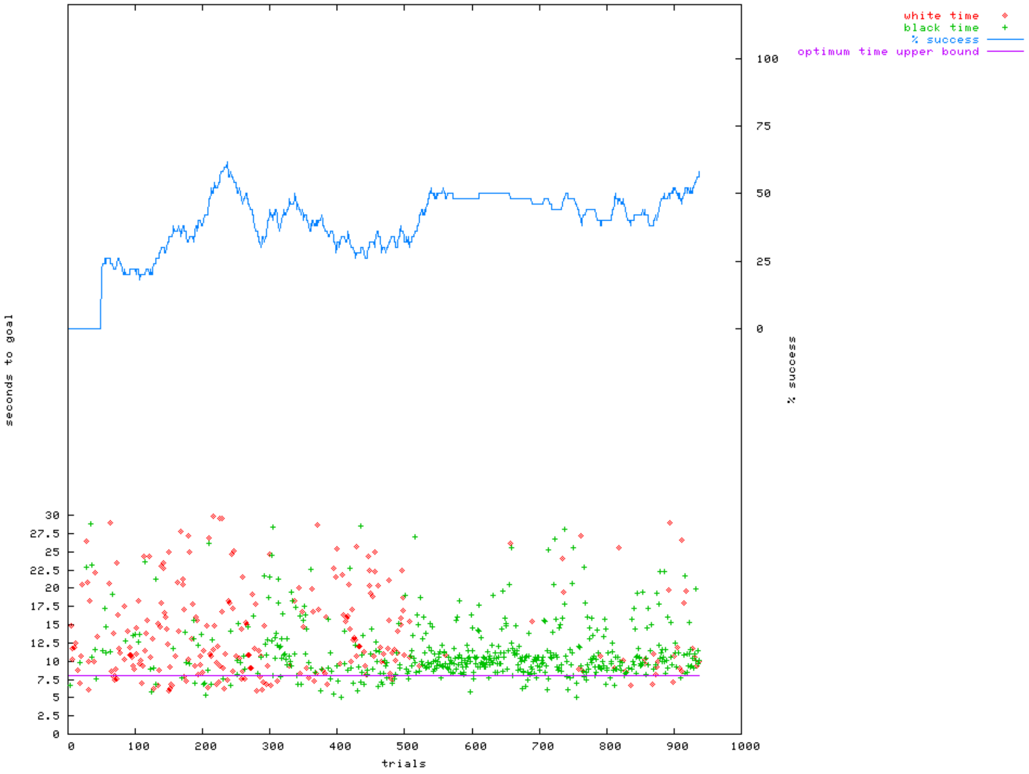


Figure 3-17 Longer run of TCS on the dual objective problem.

This was very frustrating. It may be that TCS is capable of optimally solving the dual-objective robot problem. However, as we have seen in Section 2.8.1 for a comparable problem in simulation, the high degree of epistatic linkage between the reinforcement parameters in ZCS makes it very difficult to find a set of parameters which will allow optimal behaviour. In simulation, it was easy to perform iterative searches of the parameter space to discover areas in which good parameter sets might be found. Even this painstaking approach proved unable to allow the optimal solution of the harder multi-objective problems. To carry out a similar search of the parameter space with the robot would have taken a long time; even if the robot had been run continuously, night and day, it would have taken approximately 25 years to examine the parameter space in the same detail as in simulation for a single problem, even if the runs were limited to only 200 trials!

3.5 Conclusions

Some characteristics of a useful robotic learning system have been suggested. It should not require the experimenter to decide on *a priori* discretisations of the problem space. It should be capable of determining these for itself, in order that it might be flexibly used on different hardware platforms, and in different environments. It should not require many days of training. Hurst's TCS was presented as a candidate system, which integrates a reinforcement-learning framework dealing with Semi-Markov Decision Problems into Wilson's ZCS. However, the optimal solution of a simple two objective task using TCS on a real robot could not be demonstrated.

It seems likely that TCS shares the parameter sensitivity of its parent, and it may be that optimality could be reliably achieved given much time, or luck. Therefore it would seem from these results that TCS is not suited to solving multi-objective problems of robot control.

Chapter 4 XCS

4.1 Introduction

Although results indicate that ZCS can tackle multi-objective problems if its parameters are set correctly, it is difficult to discover these settings. Indeed, in some cases it was not possible to find parameter sets that would allow the optimal solution of simple multi-step, multi-objective tasks. The majority of current LCS research uses XCS [Wilson1995], an accuracy-based classifier system, which has been shown to require less careful tuning to solve many problems optimally.

This chapter describes XCS, investigates some related work, and applies XCS to the same problems we set for ZCS in Chapter 2. The complexity of the problems is then increased, and XCS's performance is examined.

Since the aim of the work presented in this thesis is to use classifier systems to solve multi-objective control problems on a physical robot, some simple alterations to XCS are investigated that will provide a platform for this work. These bring their own problems which are examined. We will find that the performance of XCS depends on, among other factors, the chosen exploration policy, and these observations are related to current XCS theory and research.

4.2 XCS Described

XCS was first introduced in Stewart Wilson's seminal work, 'Classifier Fitness Based on Accuracy' [Wilson1995]. In the paper, he states that this work 'stemmed from dissatisfaction with the behaviour of traditional classifier systems, and the hypothesis that the short-comings were due in part to the definition of fitness'.

XCS differs from Wilson's ZCS [Wilson1994] in many ways, most importantly;

- fitness is proportionate to the *accuracy* of the classifier's prediction
- the GA does not choose parents from the population as a whole, but instead from members of action sets.

The intention behind XCS is that the system should form a set of rules that provide a complete and accurate, maximally general map of the space of state-action pairs.

XCS performs cycles of interaction and response to environmental stimulus, and unlike the 'traditional' LCS as formulated by Holland, XCS has no internal message list. In the classic implementation[13], as here [Butz2001], the environment is presented as a binary string, and this is matched to a greater or lesser extent – or not at all – by classifiers which have a 'condition' encoded as a ternary string. Covering takes place if the 'match set' thus formed, [M], is empty or the number of actions in [M] is smaller than θ_{mna}. Covering generates new classifiers with random actions, their conditions matching the current environment with characters changes to the 'don't care' symbol '#' with the probability *p#.* System parameters such as θ_{mna} and *p#* are explained in Table 4-1.

[13] All experiments here use Martin Butz's XCS classifier system implementation in C, version 1.1, available from ftp://ftp-illigal.ge.uiuc.edu/pub/src/XCS/XCS-C1.1.tar.Z

N	Maximum size of the population (sum of classifiers' numerosities – see text)
β	Learning Rate
γ	Discount Rate
α, ε_0, and ν	Used in classifier fitness updates. See text
θ_{GA}	GA threshold. The GA is applied to a set when the average time since the GA last operated is greater than this value
χ	probability of crossover
μ	probability per allele of mutation
θ_{del}	deletion threshold. If a classifier's value of *exp* is greater than θ_{del} it's fitness is included in assessing its likelihood of deletion
δ	if a classifier's fitness is less than this fraction of the mean fitness of [P] then its fitness is included in assessing its likelihood of deletion
θ_{sub}	a classifier may subsume (see text) another classifier if its value of *exp* is greater than the subsumption threshold.
p_1, ε_1, and F_1	initial values for p, ε, and F in new classifiers
θ_{mna}	specifies the minimal number of actions that must be present in [A] or covering will occur.

Table 4-1 XCS System Parameters

In addition to its condition and an associated action, each XCS classifier also records;

- *p*, the predicted value of taking this action
- *ε*, the prediction error
- *exp*, which records the number of times the classifier has been in the 'action set', [A]
- *ts*, a time-stamp that records when the classifier was last in [A] when the GA operated.
- *as*, which keeps an estimate of the average size of [A] containing this classifier.
- *n*, the 'numerosity', records the number of individuals in the population represented by this 'macroclassifier'. Macroclassifiers are treated as if they are *n* copies of identical classifiers, providing a computational advantage.
- *F*, the classifier's fitness.

Once [M] has been formed, XCS then forms a prediction $P(a_i)$ of the payoff expected from each possible action. These values are stored in the Prediction Array *PA*. Some of these values will be null if no member of [M] advocates this action. The system prediction for an action is a fitness-weighted average of the predictions of all classifiers in [M] which advocate that action.

An action is chosen from the set of actions suggested by the classifiers in [M]. Wilson suggests that ‘Many action-selection methods are possible’. Actions may be chosen *probabilistically* with the probability of selection proportional to the value of $P(a_i)$, for example via roulette selection, *randomly* without reference to the values of $P(a_i)$, or *deterministically* by picking the action with the highest value of $P(a_i)$. In typical implementations, a balance between ‘exploration’ and ‘exploitation’ is represented by performing half the trials with a random action selection policy, and the other half with a deterministic policy. This is equivalent to ε-greedy learning [Sutton1998] with ε set at 0.5.

However it is chosen, one action is selected and an action set [A] is formed. The chosen action is performed, and any external reward is returned by the environment.

The reinforcement component in XCS consists of modifying the following parameters associated with each classifier, in the following order; *exp, p, ε* and *F*. *exp* is incremented to record the number of times this classifier has appeared in an [A]. *p* and *ε* are adjusted using the Widrow-Hoff procedure with the learning rate parameter β ($0 < \beta \leq 1$) only after a classifier has been adjusted at least 1/β times, otherwise the new value is simply the average of the previous value and the current one. This technique is known as MAM (‘moyenne adaptive modifée’). That is, for classifier *j* if $exp_j >= 1/\beta$

$$p_j = p_j + \beta(P - p_j)$$

$$\varepsilon_j = \varepsilon_j + \beta(|P - p_j| - \varepsilon_j)$$

and otherwise;

$$p_j = p_j + (P - p_j) / exp_j$$

$$\varepsilon_j = \varepsilon_j + (|P - p_j| - \varepsilon_j) / exp_j$$

where P is the maximum value of the Prediction Array, discounted by multiplying by γ ($0 < \gamma \leq 1$), and with the addition of any external reward from the previous time-step. This closely resembles the Q-learning technique [Watkins1989];

$$Q(s_t, a_t) = Q(s_t, a_t) + \beta (R + \gamma \max Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

where $Q(s_t, a_t)$ is the payoff predicted under the hypothesis that the agent performs action a_t in state $s_{t,}$ and thereafter always selects the action predicting the highest payoff, γ is the discount rate, and R is the reward received for performing action a_t in state $s_{t,}$ In contrast to Tabular Q-learning, where a Q value is derived for every state-action pair in the environment, an LCS allows the generation of rules that generalize over many states. In Reinforcement Learning, function approximation techniques such as artificial neural networks are used for similar reasons; see [Sutton1998] for an overview.

In the final step of the reinforcement process, each classifier in the set has its fitness F adjusted. This is performed in three stages;

- Each rule has its accuracy K_j determined:

  $$K_j = \alpha (\varepsilon_{j / } \varepsilon_0)^{-v} \text{ for } \varepsilon_{j >} \varepsilon_0 \text{, otherwise } K_j \leftarrow 1.$$

- A relative accuracy K'_j is determined for each classifier by dividing by the sum of accuracies of classifiers in the set. Numerosity must be accounted for here.
- The Widrow-Hoff delta rule is applied with the learning rate β:

  $$F_j = F_j + \beta (K'_j - F_j)$$

As noted above, the second main area in which XCS differs from ZCS is in the application of the GA. Wilson was motivated to develop XCS by the belief that ZCS failed to achieve optimality due to the emergence of over-general rules; rules which are good in one niche and achieve a high fitness value are chosen over more optimal rules in a different niche where they also match, but in which they are of lower utility. Wilson attempted to address this problem by the use of accuracy as a basis for fitness, and also through restricting the GA to choose parents from within [M].

In the version of XCS used in this chapter, the GA is restricted in its choice of parents to [A] rather than [M], as first detailed in [Wilson1998]. By restricting the GA to [A], there is exploration of the space of possible generalizations containing a specific state-action pair. Since classifiers that participate in more niches (and are accurate) are rewarded more often and have more opportunities to replicate and search the space of generalizations through the genetic operators, this introduces a pressure towards the generation of **general** classifiers *for each state-action pair explored by the system.*

The 'triggered-niche' GA used by XCS in this implementation works in the following way. Each classifier has an associated measure of the number of time steps since it was last involved in an application of the GA. The GA is invoked if the average time period since the classifiers currently in [A] were last subject to the GA is greater than a threshold value θ_{GA}. Two parents are selected by a fitness-proportionate method (roulette selection) and after possible two-point crossover and mutation, are inserted into the population.

Classifiers are selected for deletion from the population as a whole [P], while parents are selected from [A]. If the sum of the numerosities is less than the preset maximum *N*, no deletion takes place. Otherwise, a classifier is chosen to be deleted (if it is a macroclassifier, its numerosity is decreased by 1) by roulette selection with increasing probability on the strength of its ‘deletion vote’. The latter is calculated based upon numerosity and the classifier’s action set size estimate. Additionally, if the classifier is sufficiently experienced i.e. $exp > \theta_{del}$, and its fitness is less than δ multiplied by the average fitness in [P], its deletion vote is increased in inverse proportion to its fitness.

There are two forms of ‘subsumption’ in XCS that contribute to computational efficiency at the expense of a decrease in diversity in the population. In ‘GA subsumption’ an offspring that has a condition already represented by a more general and experienced parent is not added to the system, but the parent’s numerosity is incremented. In ‘Action Set subsumption’, the most general classifier in [A] is found, and all other classifiers in [A] are tested against it to see whether it subsumes their condition. If it does, they are discarded and the most general classifier has its numerosity increased by one as before.

Other work using XCS includes that of Butz et al. [Butz2003b] in which they examine the pressures that influence generality in XCS. They recognise;

- Set Pressure: since classifiers are more likely to be reproduced the more often they are in [A], but classifiers are deleted from [P] as a whole, there is an intrinsic pressure towards generality.
- Mutation Pressure: mutation changes alleles, either at random (‘free mutation’), or while maintaining a match to the current input (‘niche mutation’). This provides a pressure depending on the frequency of the GA and the mutation rate towards a specificity of 0.66 or 0.5 for the two types of mutation, respectively.
- Deletion Pressure: deletion depends upon *as* and fitness. In the absence of other pressures, the average specificity of deleted classifiers will approximate to the average specificity of [P].

- Subsumption Pressure: subsumption provides an explicit pressure towards generality.
- Fitness Pressure. In general, since fitness is dependent on accuracy, the effect of fitness will be to inhibit over-general rules that are inaccurate in some niches, thereby producing a pressure towards the specific.

They also note two 'challenges' that XCS must overcome in order to learn a problem. Firstly, if cover is invoked too often, classifiers may be deleted before their true value can be assessed The parameter θ_{del} was introduced to minimise this problem, first noted in [Kovacs1999]. Whether this happens will also depend on the complexity of the problem, the setting of *p#*, and the value of *N*. They term this the '*covering challenge*'.

Secondly, since XCS uses accuracy-based fitness, [P] must contain enough *specificity* – building blocks with matching characters that are not wildcards - to allow the production of accurate classifiers. The impact of this '*schema challenge*' will clearly depend on the extent to which the problem space can be solved by general rules - how rugged is the problem landscape - and also upon the settings of *p#* and *N* as before.

In [Butz2003a], Butz et al. extend this examination of the pressures on XCS to include time, and also examine the effect of population size *N*. They suggest that 'the necessary specificity actually decreases with increasing population sizes', although their main finding is the reverse of this. They derive equations for the 'Reproductive Opportunity Bound' (ROP) in XCS, based on the probability that a suitably specific classifier will be generated by mutation, and subsequently maintained in the population by reproduction, and not deleted. It should be noted that their theoretical treatment assumes that all states in the environment are sampled with equal frequency.

The results presented in the following sections of this chapter show that it is not always the case that the necessary specificity of the population decreases with increasing values of *N*, suggesting that the ROP is influenced by the action selection scheme. It appears that there is another challenge to XCS's success, which could be termed the '*niche resource challenge*'.

Many methods could be used to select an action suggested by the classifiers forming [M]. In most multi-step experiments that have been described in the literature, a balance has been enforced between 'exploration' and 'exploitation' by simply performing half the trials using a random action selection policy, and half using a deterministic policy. Results are presented based purely upon the performance of the deterministic trials [e.g. Wilson1995].

Since it is the intention in these investigations to aim always at the use of LCS for multi-objective problems performed by physical agents in the real world, this presents some minor problems. Although in simulated problems this simple distinction between random exploration and deterministic exploitation is a valid approach, one must doubt the utility of any agent that performs randomly 50% of the time.

Thus the experiments that are presented below first demonstrate the performance of XCS using the traditional 50/50 random exploration, and then investigate XCS' performance using roulette-wheel selection in the prediction array during the exploration trials. To enable easy comparison with results in the literature, results are still presented based upon only the deterministic trials.

An adaptive ε-greedy parameter could be used, adjusting ε according to some measure of error in [P] or more likely [A]. Wilson [Wilson1994] mentions some possible methods of self-adapting the balance of exploration and exploitation. However, as the values of fitness are updated, roulette-wheel selection can be said to self-anneal so that the system gradually converges upon a mainly exploitative policy while still allowing some exploration to take place.

4.3 Sequential Multi-objective problems

The performance of XCS is first examined in two sequential multi-objective tasks. These take place in adaptations of the familiar Woods1 grid-world [Wilson1994]. In these, and all following experiments with XCS, all results are the average of 10 experiments, presented as the running average of the last 50 deterministic exploit trials as in [Wilson1995].

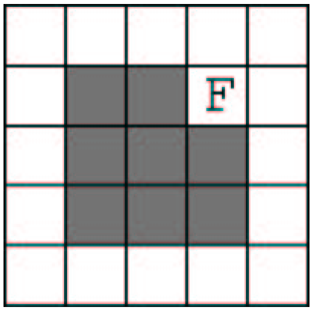


Figure 4-1 The Woods1 Environment.

Woods1 is a toroid comprised of 25 squares. A virtual agent, or ‘animat’ (Wilson, ibid.), moves between the squares under the control of the learning algorithm. Movement is possible from a square to any of the eight adjoining squares, except where this contains a ‘rock’ (shown in Figure 4-1 as a grey square). A trial starts in an empty square chosen at random, and finishes when the animat reaches the goal state, here marked with ‘F’ for ‘food’. In the experiments described here, three[14] characters are used to describe each state in the environment, and the environment is presented to the classifier system as a binary string representing the state of the eight cells that surround the animat’s current position, from ‘North’ clockwise to ‘North-West’. The condition of a classifier has an equal number of characters to the environmental representation. Three characters, allowing the eight possible directions to be represented, again encode the action. In the Woods1 environment, the reward

[14] Although only two characters are required in to encode the states in the Woods1 environment, three are necessary as the number of different states increases in the further multiobjective problems reported here. We have used the same encoding throughout.

given upon reaching the goal is 1000, and the optimum average path from start to finish is 1.7 steps.

4.3.1.1 Key and Door

In the first extension of the Woods1 environment, the animat must visit a state in the environment before it reaches the goal. If it does not first visit this state, the trial will not terminate. This can be thought of as getting a 'key' to open a 'door' to gain reward. In order to allow XCS to 'remember' whether the animat has previously visited the 'key' state, an extra character is added to the representation of the environment, this character being set from its initial state of zero to one when the animat visits the 'key'. This can be matched by a concomitant extra character in a classifier's condition. In Figure 4-2, the 'key' is shown as 'K', and the reward state is indicated as 'F' (food).

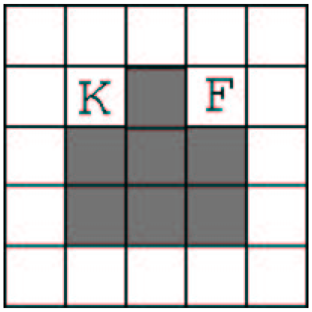


Figure 4-2 Woods1 'Key and Door'

The average optimum number of steps from any position in the environment to the goal state, going by way of the 'key', is 3.7. The results presented below in

Figure 4-3 and Figure 4-4 show the performance of XCS with a population size (*N*) 1600 and 3200, and random exploration. Figure 4-5 and Figure 4-6 show the same treatments using roulette-wheel selection.

Other parameters are;

Table 4-2 Common parameters used in all XCS experiments

(unless otherwise stated)

β	0.2	Learning Rate
γ	0.71	Discount Rate
θ_{GA}	25	GA Threshold
ε_0	0.01	used in fitness calculations
α	0.1	
v	0.1	
χ	0.8	probability of crossover in GA
μ	0.01	probability of mutation per character in GA
θ_{del}	20	if exp > θ_{del} *AND* fitness < δ * (mean fitness in[P]), deletion vote increases in inverse proportion to fitness.
δ	0.1	
p#	0.33	likelihood that a classifier created by cover will have a '#' at a character.
p_I	10	initialisation values for new classifier
ε_I	0	
F_I	10	
θ_{sub}	20	a classifier can subsume another if exp > θ_{sub}
θ_{mna}	8	if number of actions in [M] is less than θ_{mna}, cover will take place

Only GA subsumption is performed.

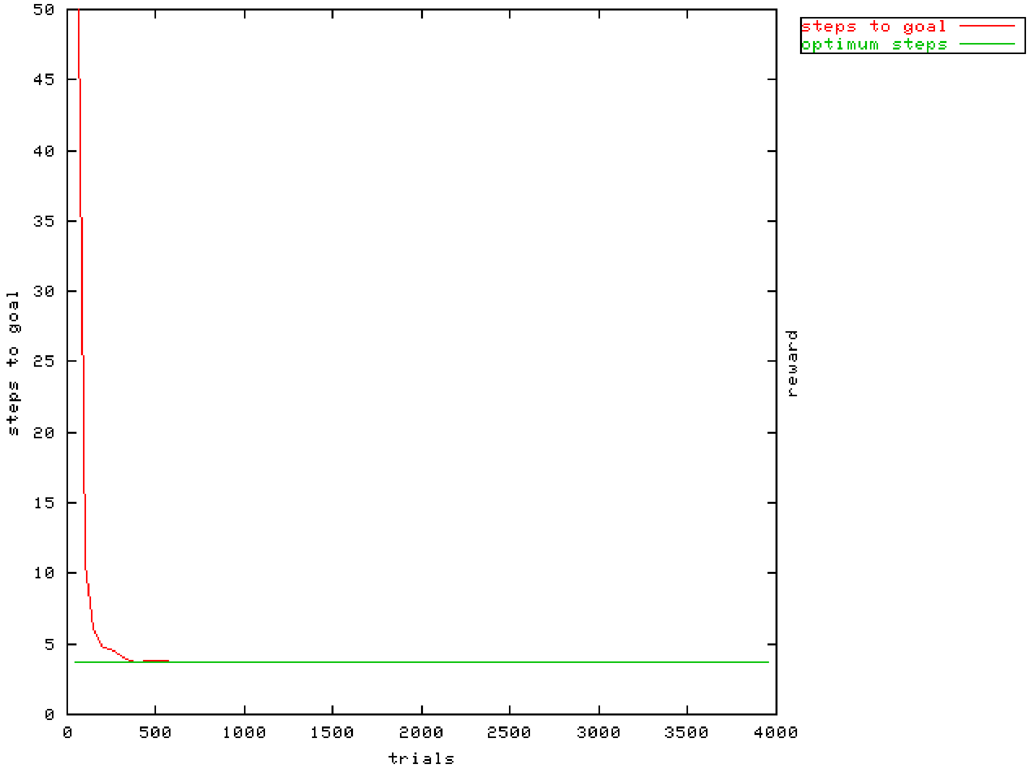


Figure 4-3 Key and Door: N=1600, random exploration

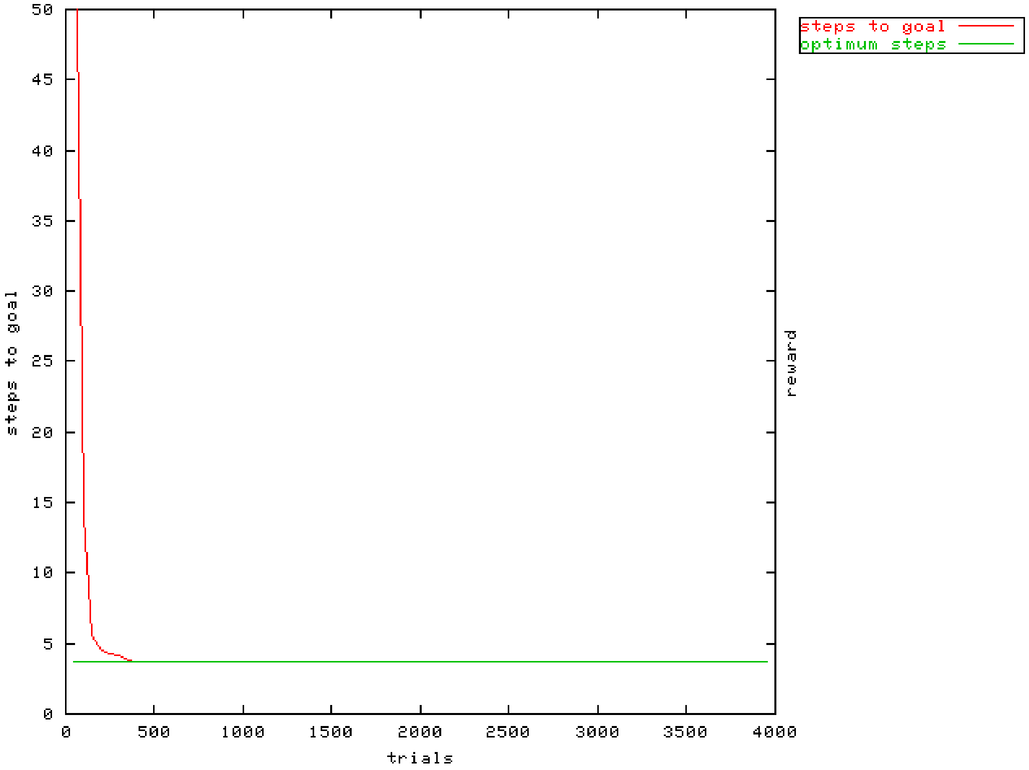


Figure 4-4 Key and Door: N=3200, random exploration

As can be seen, in all treatments XCS achieves optimal performance. However, it would appear as though it takes longer to converge to an optimal solution with the roulette action selection mechanism; with *N*=1600 and using random exploration, XCS achieves optimality after about 500 exploit trials, whereas using roulette exploration requires almost double this number of trials with the same value of *N*. Increasing *N* is required to allow roulette selection to perform as well; here a doubling of *N*.

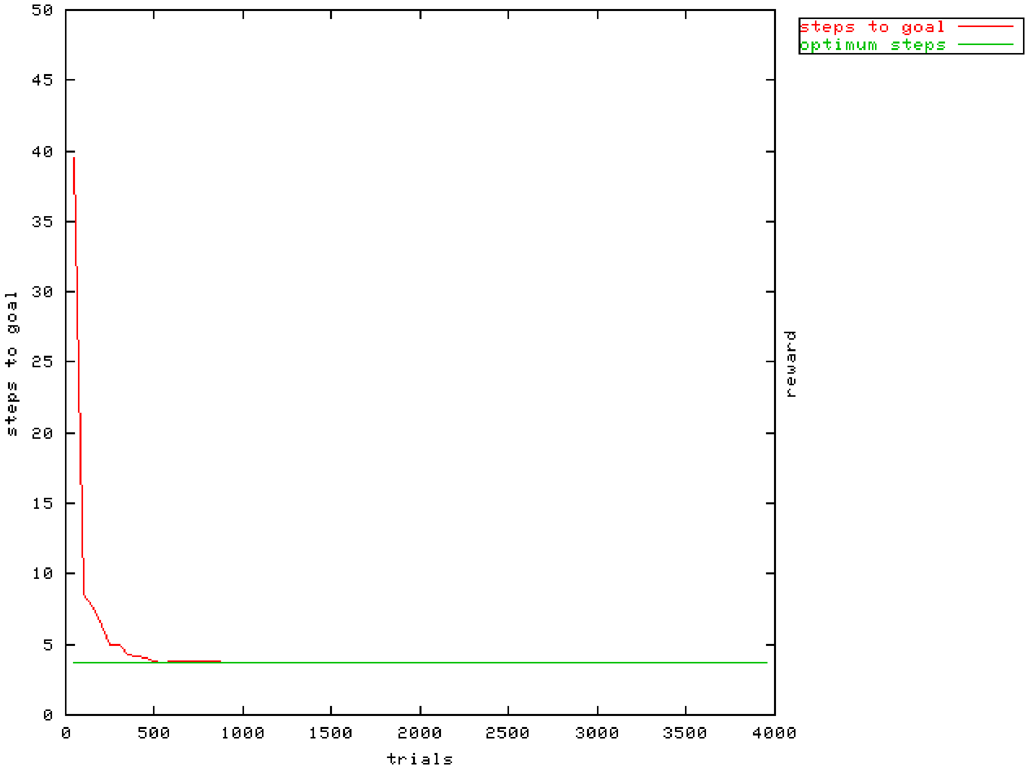


Figure 4-5 Key and Door: *N*=1600, roulette exploration

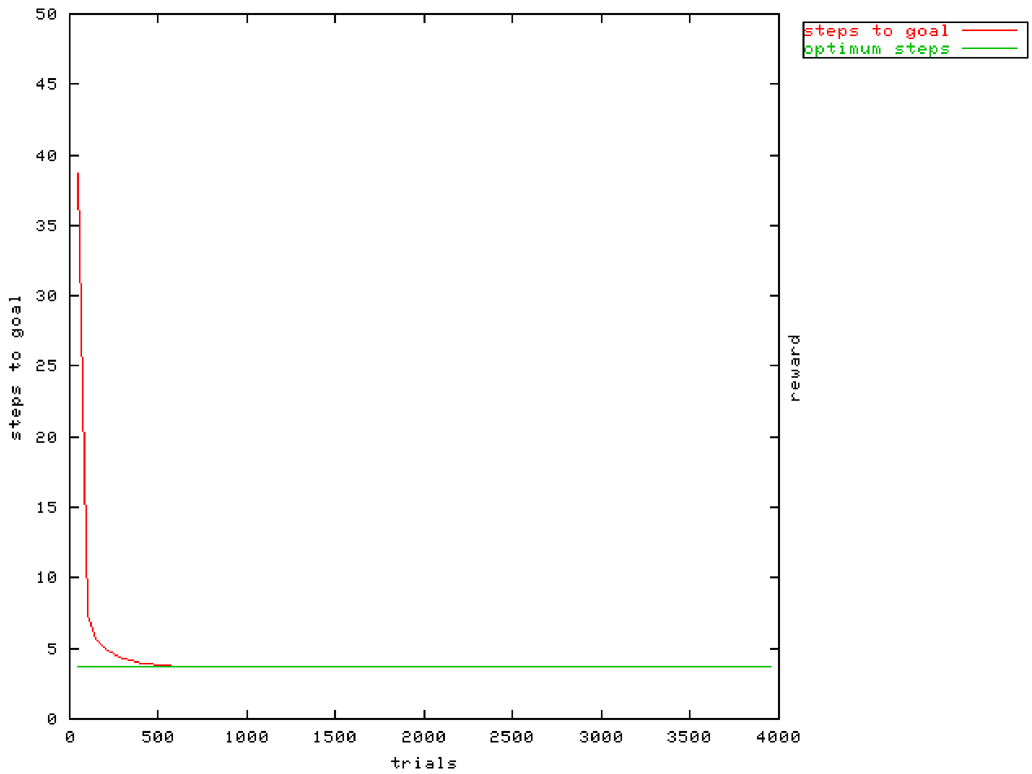


Figure 4-6 Key and Door: *N*=3200, roulette exploration

4.3.1.2 Carry the Flag

Of course, the above is a rather trivial problem. The animat can only achieve its reward and terminate the trial if it has first visited the key state. A slightly more complicated variant is provided by the same environment, in which the 'door' is always open. However, the reward gained is dependent on whether the animat has previously visited the 'key' state, being 1000 if it has, and otherwise 1. Once again, an extra character in both environment and classifier condition allows XCS to 'remember' whether the animat has previously visited the 'key' state.

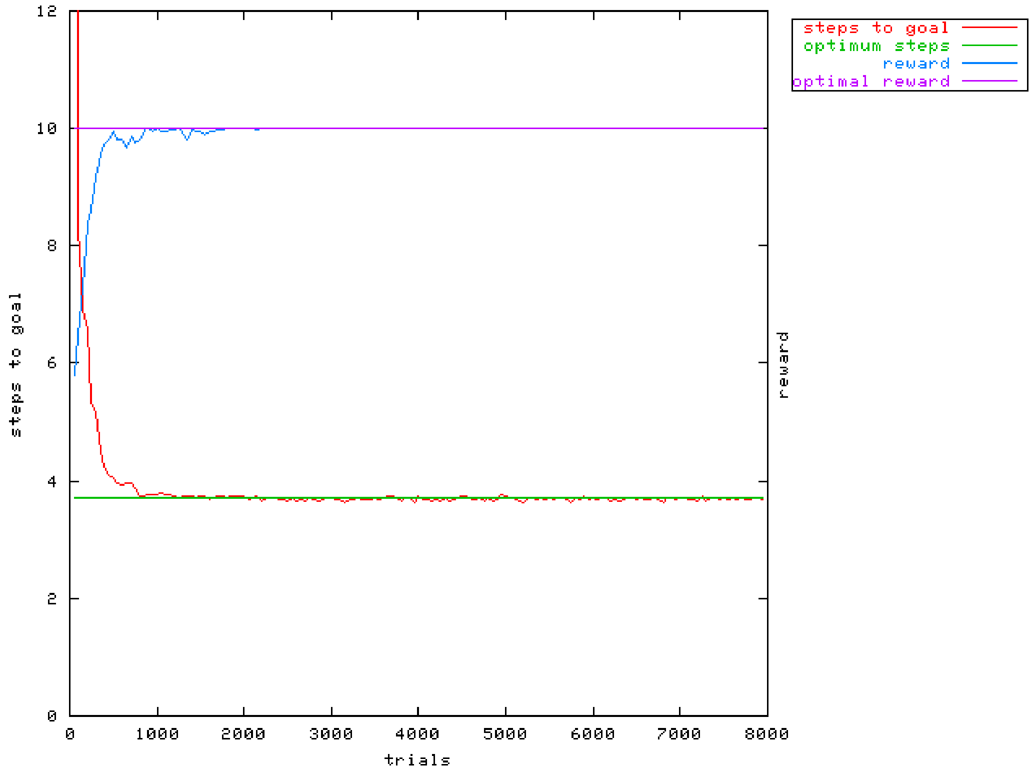


Figure 4-7 'Carry the Flag', *N*=1600, random exploration

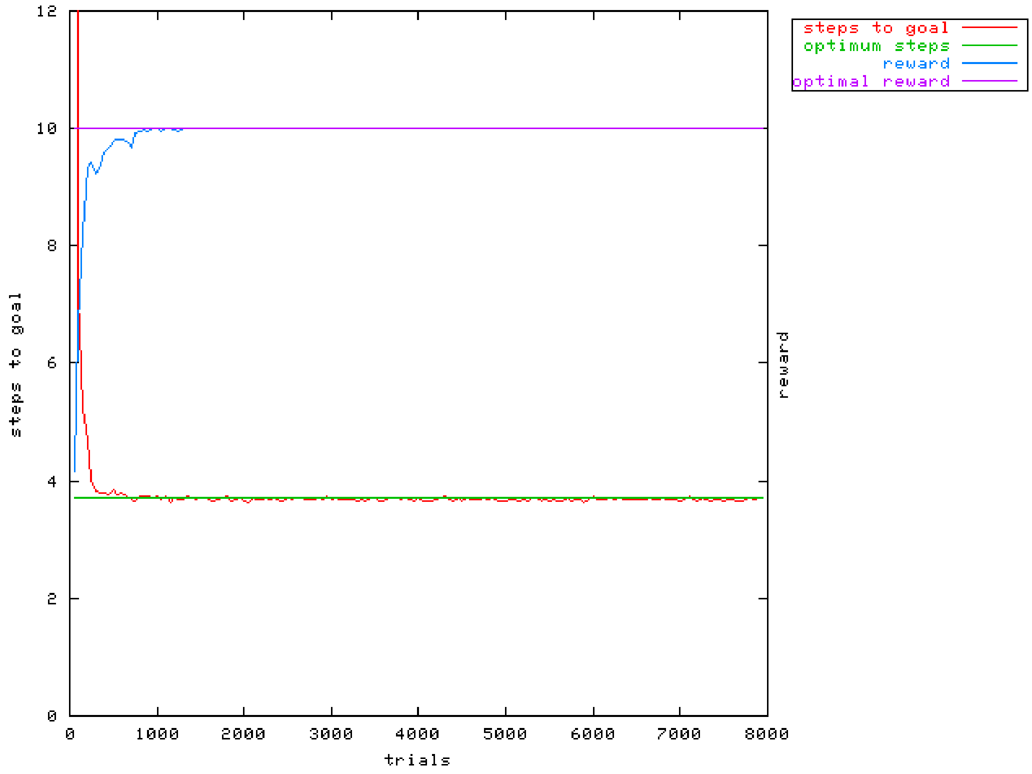


Figure 4-8 'Carry the Flag', *N*=3200, random exploration

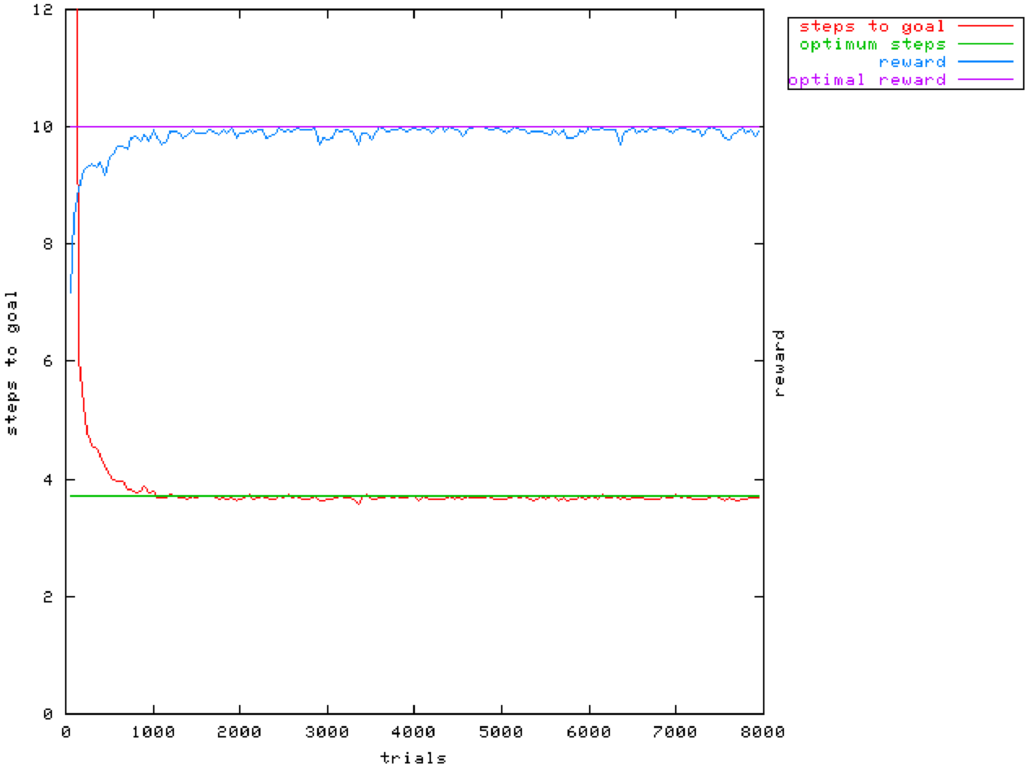


Figure 4-9 'Carry the Flag', *N*=1600, roulette exploration

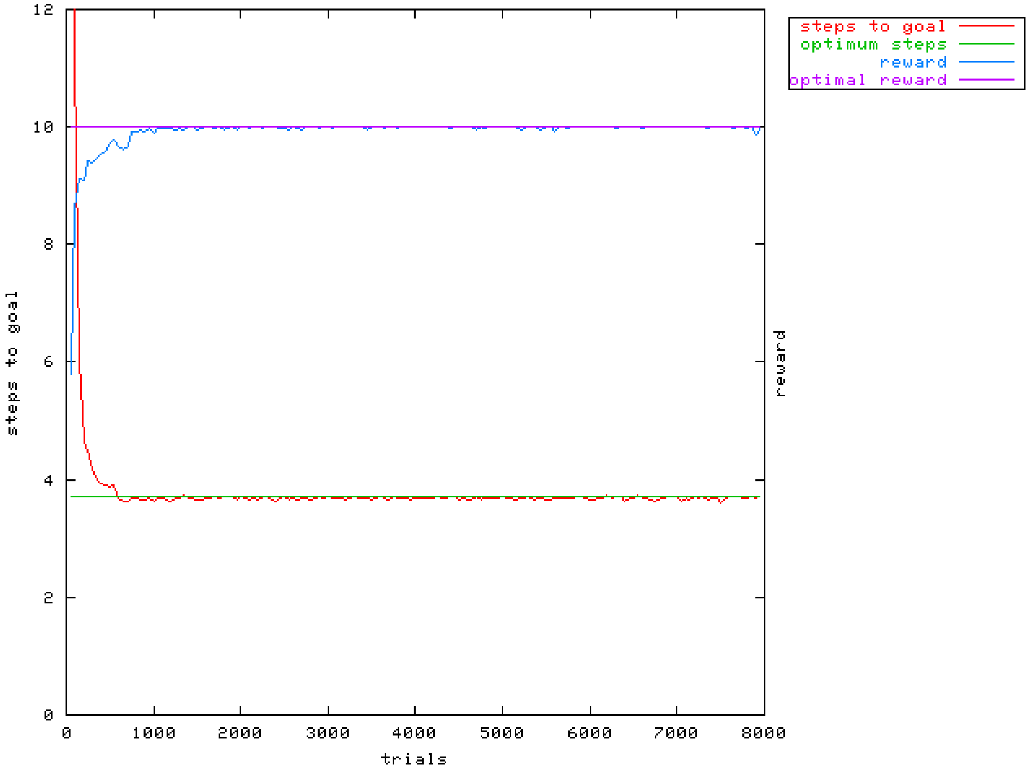


Figure 4-10 'Carry the Flag', *N*=3200, roulette exploration

Figure 4-7 -Figure 4-10 show the performance of XCS on this second sequential multi-objective task. With both roulette-wheel and random exploration, there is a slight reduction in the time needed to achieve optimal steps to goal as the population size is increased from 1600 to 3200. Once again, we also see that random exploration achieves optimal performance faster than with roulette-wheel exploration. Note also that where *N*=1600, the treatment with random exploration clearly achieves a better average reward (1000 ± 0.00 (mean ± std. dev.)) than the treatment using roulette-wheel exploration (986 ± 9.51 (mean ± std. dev.)), implying that the map of predicted payoffs is less accurate in the latter.

4.4 Concurrent Multi-objective problems

4.4.1 Two objective problems

We have seen that XCS is capable of optimal performance on the simple sequential two-objective problems using both random and roulette-wheel exploration policies. It is more interesting to consider the problems faced by a learner that has to juggle multiple simultaneous objectives. In order to do this, an alteration of the previous Woods1 environment is used that not only has a ‘food’ goal, but also has another goal labelled ‘energy’. In a similar fashion to the second sequential multi-objective task, the environment is presented to the classifiers with an additional character, which is set to one if the animat’s internal ‘energy level’ is higher than 0.5, and otherwise set to zero. The internal energy level can vary between 0 and 1. The classifiers once again have an additional character in the condition that allows them to match this extension of the environment. A trial is terminated and reward is given when the animat reaches either of the goal states.

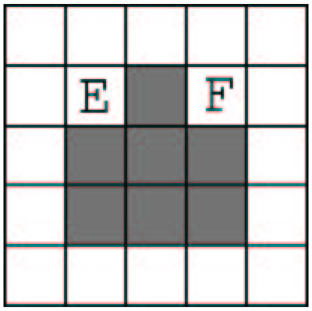


Figure 4-11 The Woods1e environment with two goals

4.4.1.1 Stepwise reward, no cost of movement(Woods1e-type1)

In the first version of the Woods1e problem there is no cost of movement. Each trial starts with the animat's internal energy level initialised at random in the range [0, 1]. The reward function is step-wise, giving an external reward of 1000 in the case that the animat arrives at the energy goal when its energy level is lower than or equal to 0.5, the reward being otherwise 1. Conversely, the animat receives a reward of 1000 if it arrives at the food goal when its energy level is higher than 0.5, the reward being otherwise 1. This problem and the Woods1e environment were first reported in [Bull2002b]. The optimal average reward that can be attained is 1000, and the optimal average steps to either goal state is 1.7.

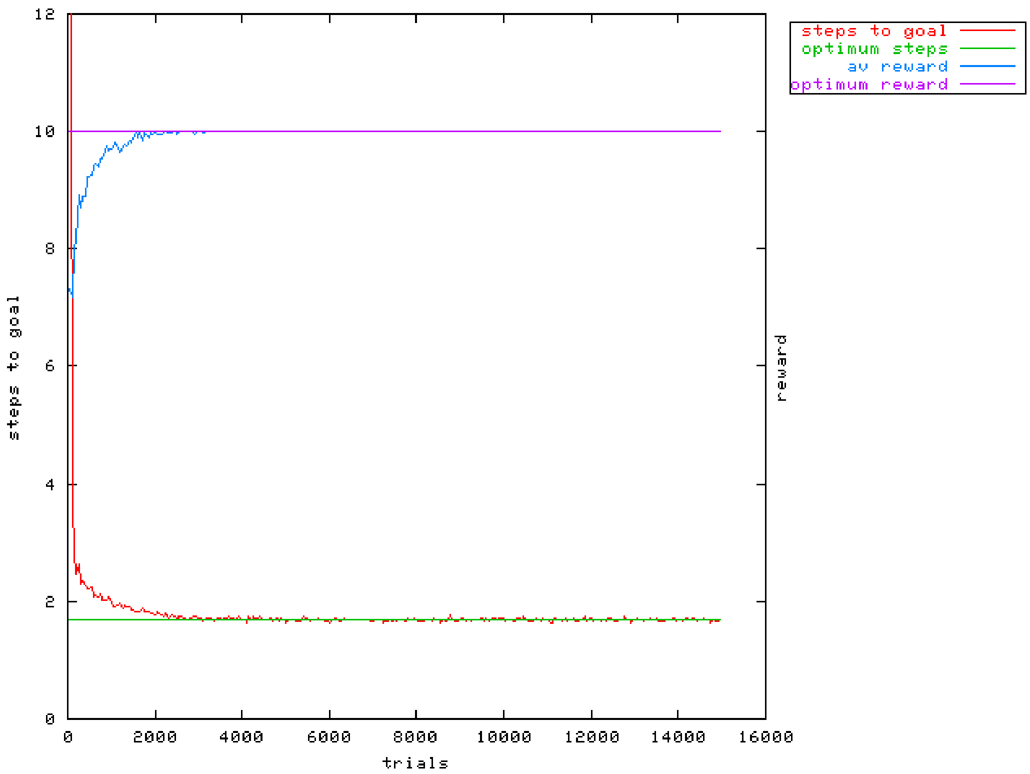


Figure 4-12 Woods1e-type1. *N*=1600, random exploration

Figure 4-12 shows XCS achieving optimal performance with *N*=1600 using random exploration. In Figure 4-13 it will be seen that using roulette exploration, performance is slightly worse in terms of the average reward achieved (random 1000 ± 0, roulette 993 ± 3.69 (mean ± std. dev)). Increasing the population size allows XCS to address this problem using roulette exploration (see Figure 4-14), again indicating the significance of *N* as seen in the sequential problems above.

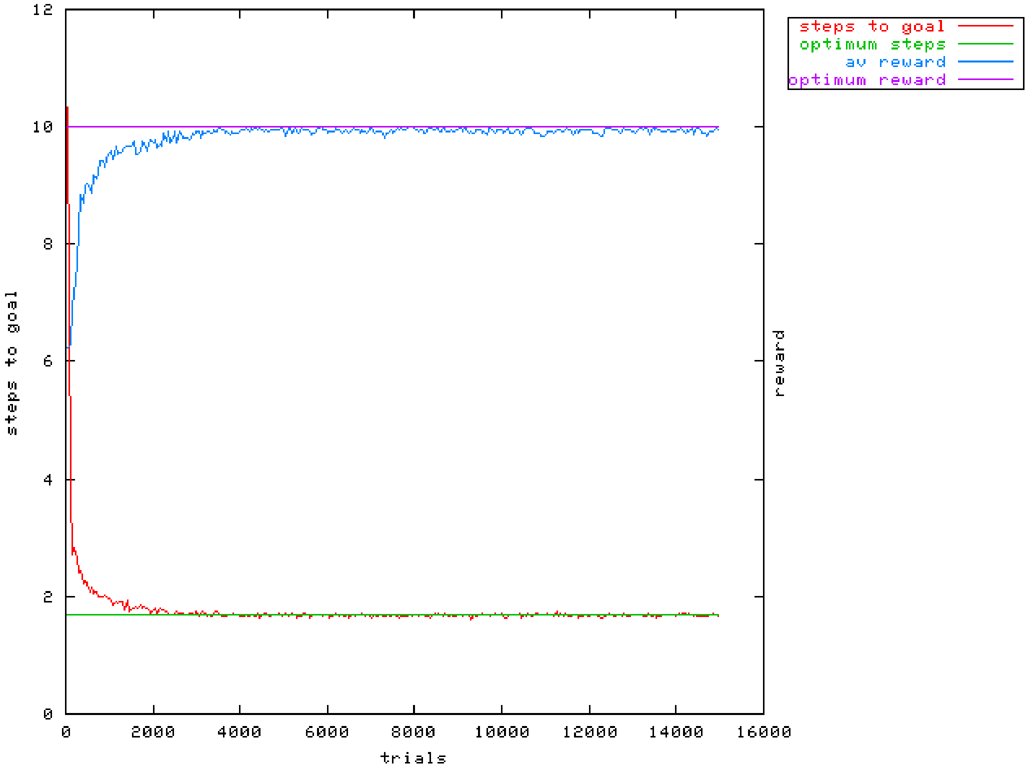


Figure 4-13 Woods1e-type1.   *N*=1600, roulette exploration

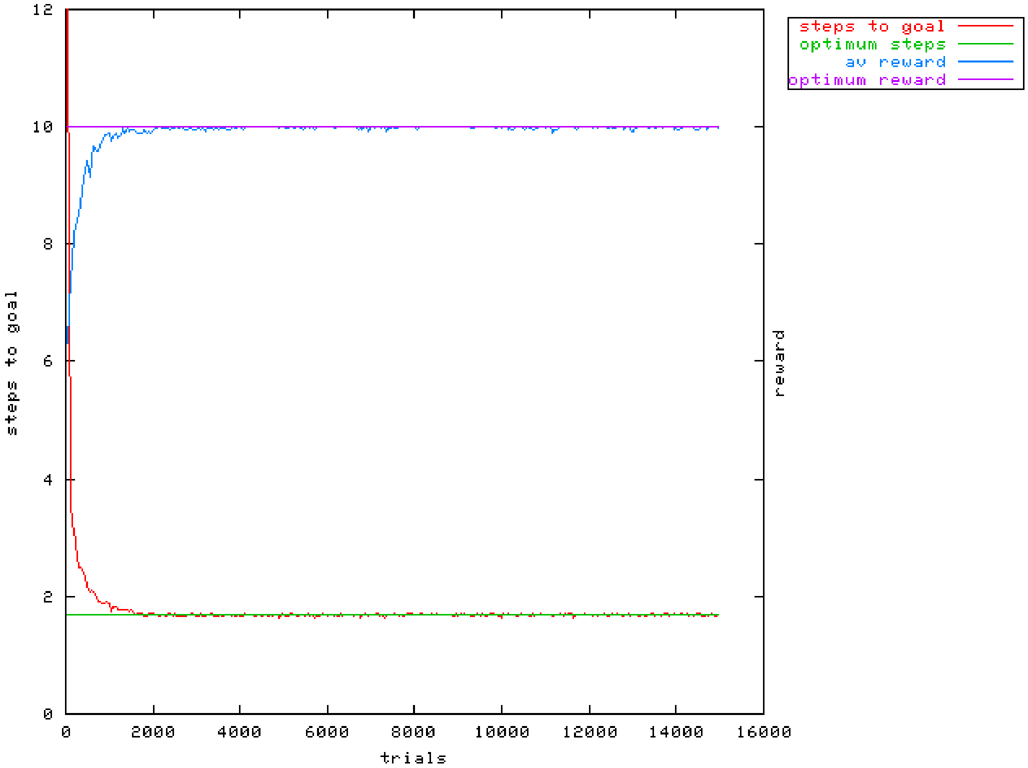


Figure 4-14 Woods1e-type1.   *N*=3200, roulette exploration

In Figure 4-15 - Figure 4-17 we see how average steps to goal, average reward achieved, and average performance vary in response to changing values of learning rate and discount rate, using the random exploration policy. Performance is measured as before (Section 2.8.1).

It is apparent that XCS is largely insensitive to the setting of learning rate and discount rate, apart from at extreme settings of γ and β. Indeed, optimal (or very nearly optimal) behaviour appears to be the norm rather than the exception.

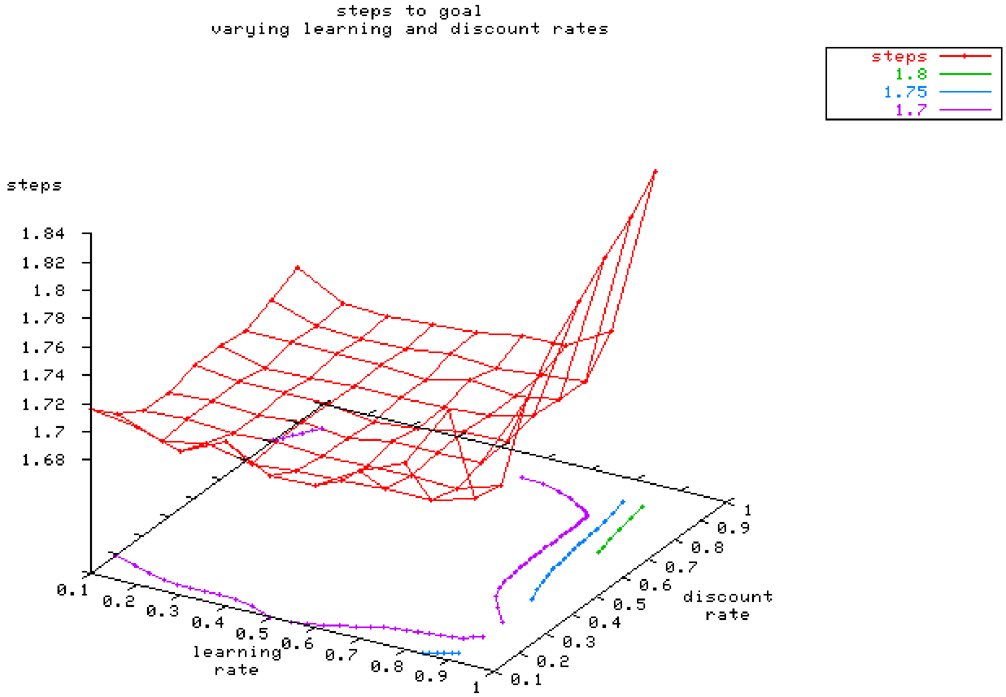


Figure 4-15 Exploration of steps to goal with varying learning- and discount- rate.

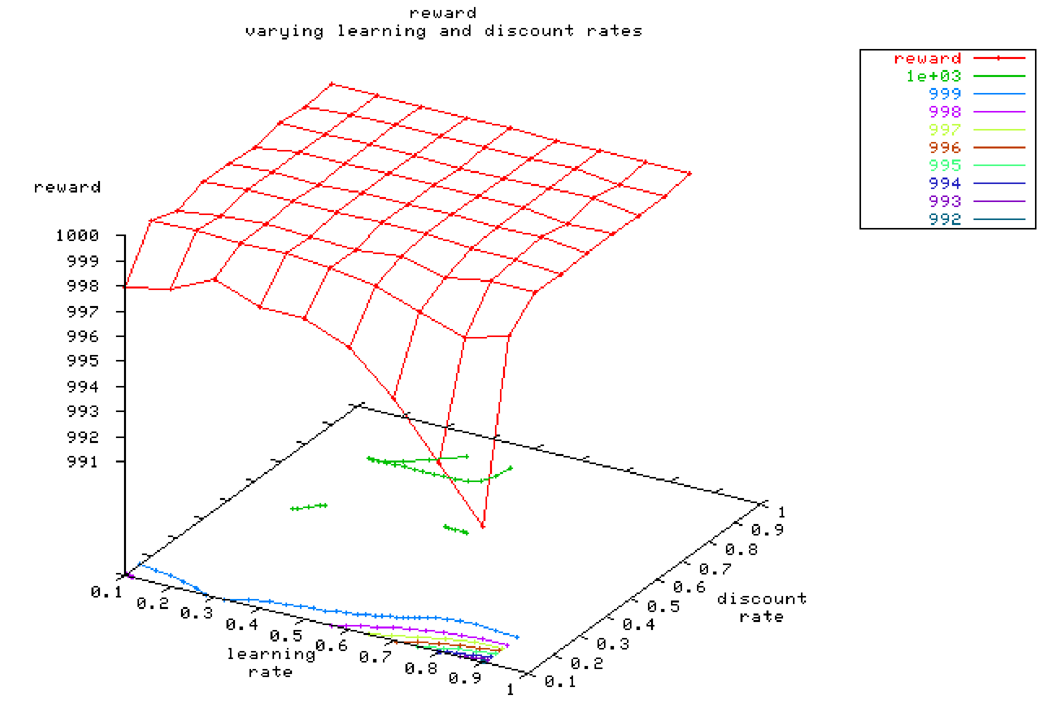


Figure 4-16 Exploration of reward gained with varying learning- and discount- rate.

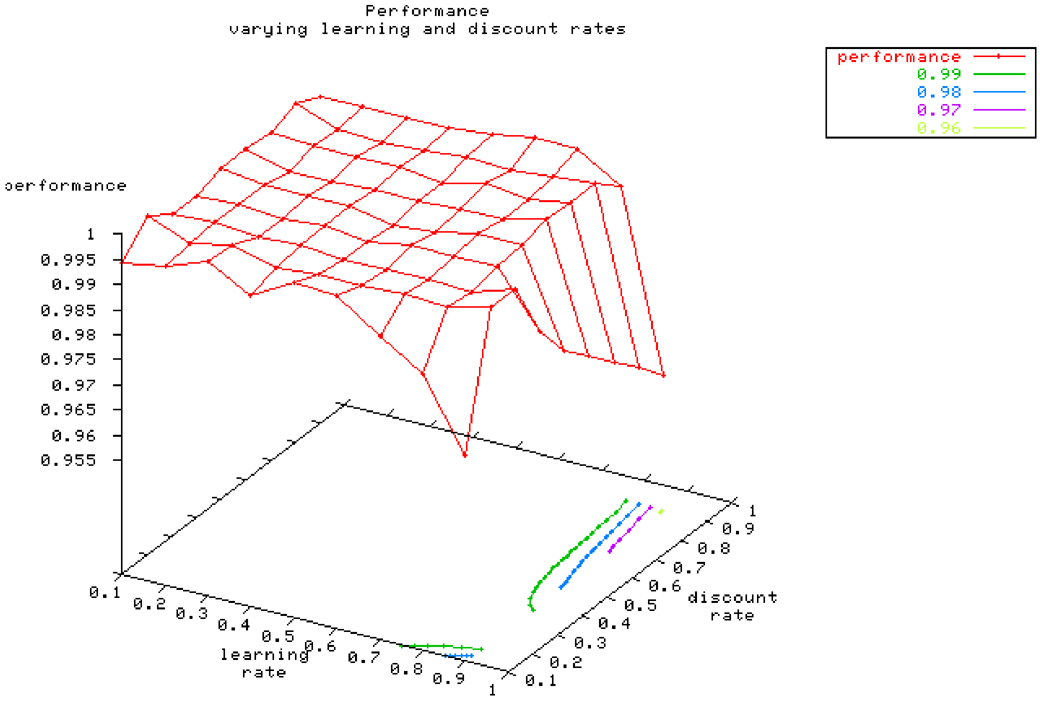


Figure 4-17 Exploration of performance with varying learning- and discount- rate.

4.4.1.2 Stepwise reward, dynamic cost(Woods1e-type2)

Once again, a cost of 0.01 ‘energy points’ as introduced for each move made by the animat. This cost is deducted from the animat’s internal energy level *after* it has been assessed whether the most recent action has brought an external reward, and if so, what size the reward should be. (This is important; if the animat is charged for moving *before* the potential reward is assessed, the environment is made ambiguous in those trials where the internal energy level is 0.51. The animat moves to the ‘food’ state ‘expecting’ a high reward of since the energy level is above 0.5, but this decision causes it to receive a low reward.) All other details remain the same as in the case with no cost of movement.

Figure 4-18 and Figure 4-19 show two treatments with random exploration, while Figure 4-20 and Figure 4-21 show the same treatments using roulette exploration. Once again we see that it is necessary to increase the population size with roulette exploration to achieve similar performance as that attained with random exploration.

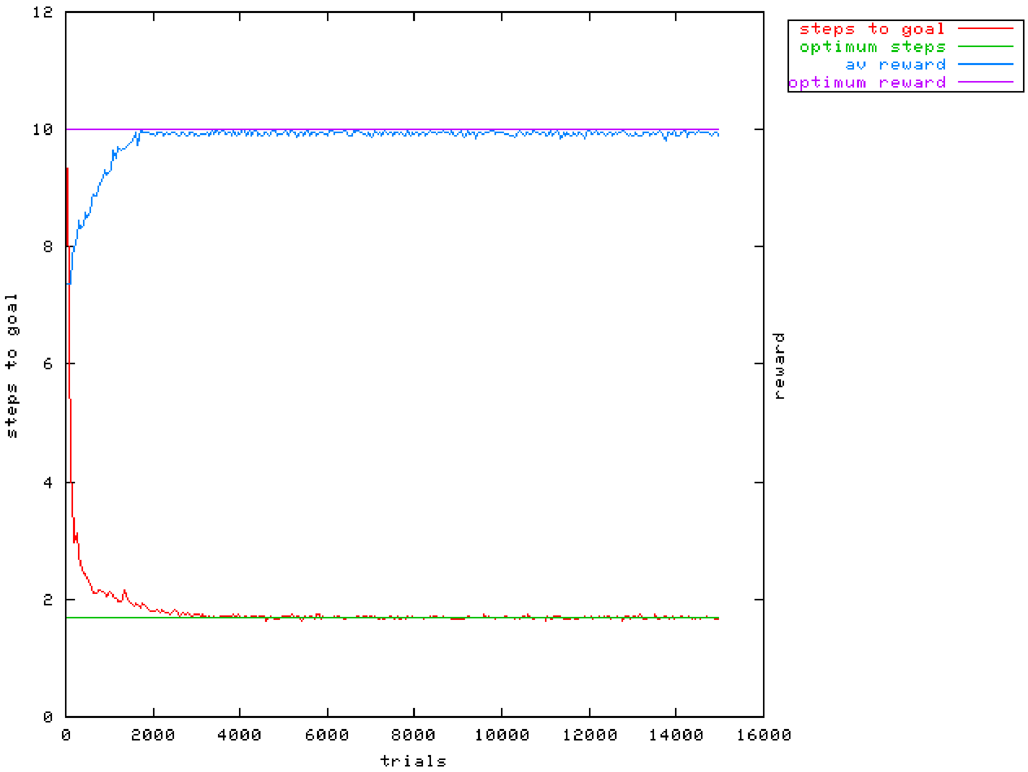


Figure 4-18 Woods1e-type2. *N*=1600, random exploration.

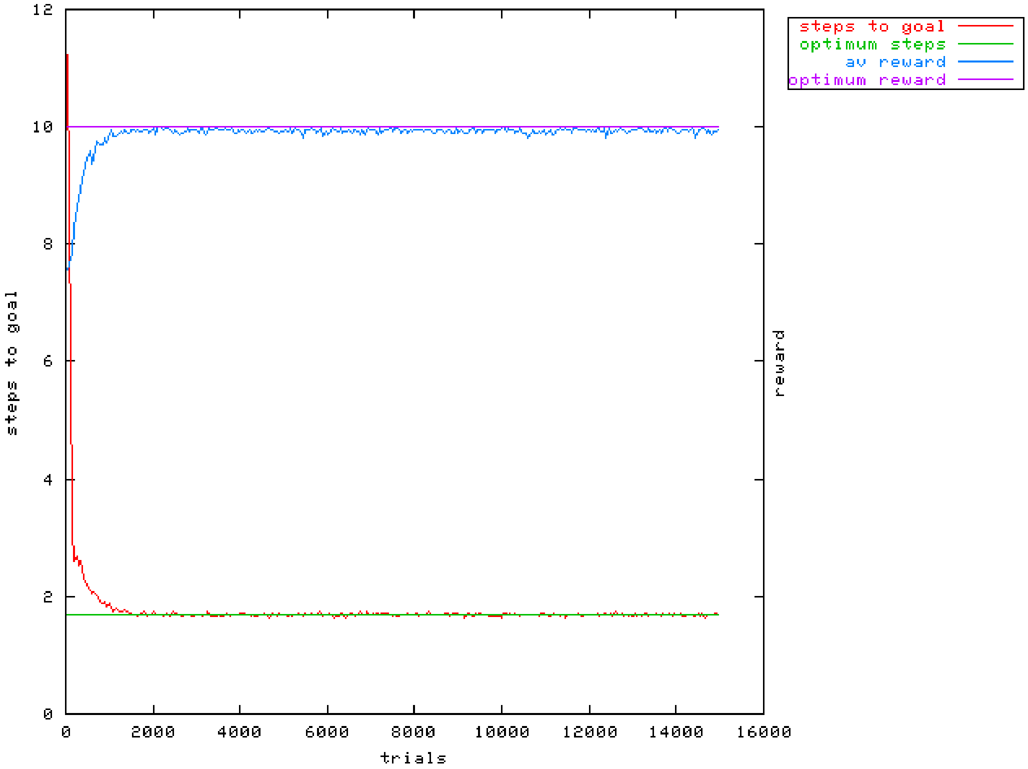


Figure 4-19 Woods1e-type2. *N*=3200, random exploration.

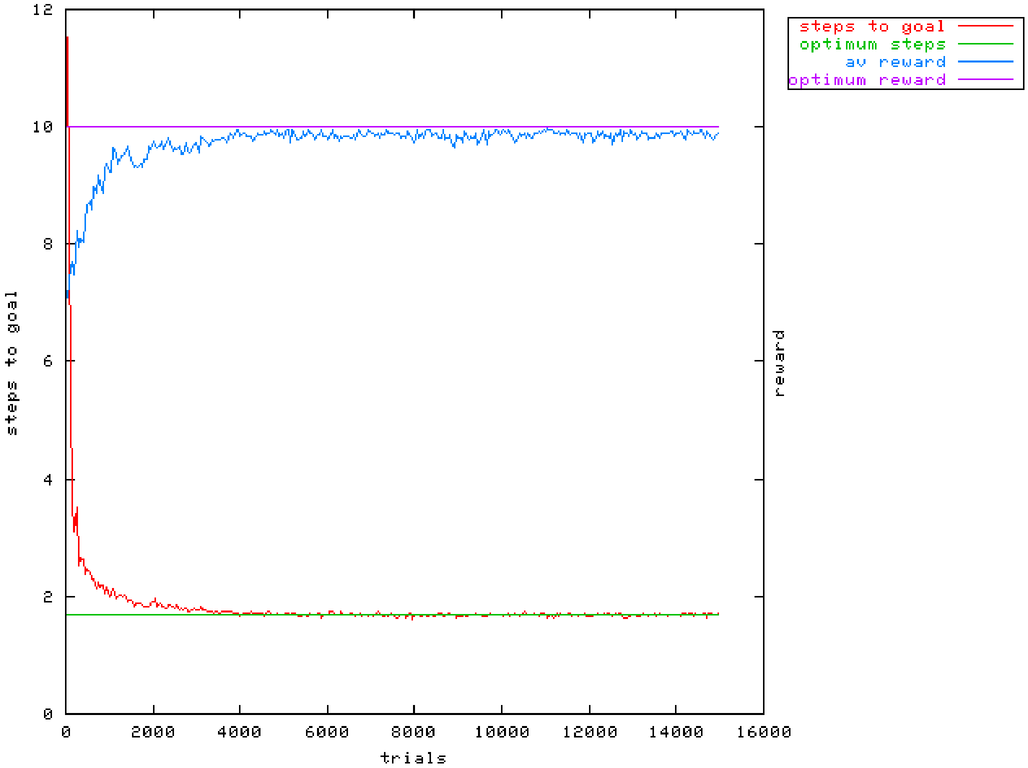


Figure 4-20 Woods1e-type2. *N*=1600, roulette exploration.

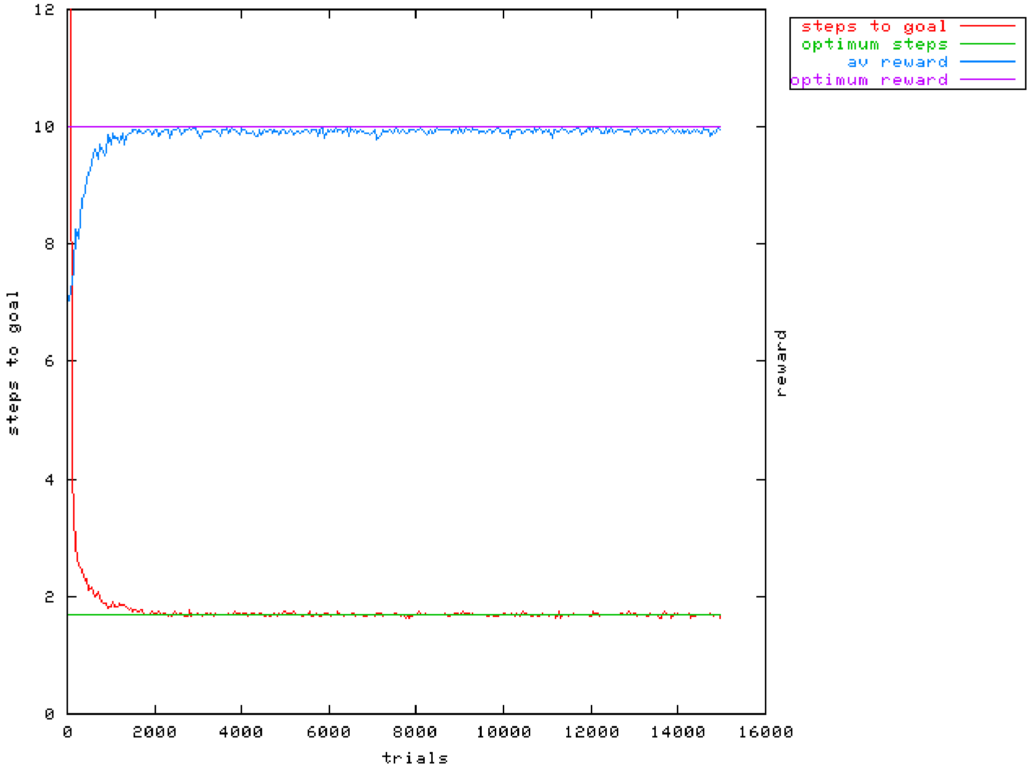


Figure 4-21 Woods1e-type2. *N*=3200, roulette exploration.

	t Stat	t Critical two-tail
Steps to Goal	0.191	0.850
Reward	0.125	0.902

Table 4-3 Two-Tailed Student t-test, Roulette N=3200, Random N=1600

As shown in Table 4-3, the difference between the treatments is not significant at the 95% confidence level either in terms of the steps to goal or the reward achieved.

4.4.1.3 Continuous reward, dynamic cost(Woods1e-type3)

In a further effort to make the simulation more like the problem faced by a real robot, XCS is now set the task of finding an optimal solution to the Woods1e problem when its energy level is altered by its movements, and where the reward received upon arriving at one or other goal state is directly proportional to the energy level, rather than varying in a step-wise fashion according to the energy level as has hitherto been the case.

At the ‘energy’ goal	Reward = 1000.*e*
At the ‘food’ goal	Reward = 1000(1-*e*)

Where *e* is the animat’s internal energy level that varies between zero and one. As before this internal energy level is set randomly at the start of each trial, such that half the trials will start with $e < 0.5$. As in the previous experiment, a cost of 0.01 ‘energy points’ is deducted for each move made by the animat in the grid-world. The animat’s energy level is not allowed to go below zero. As in the preceding experiments, the real value of the internal energy level is hidden from XCS, and is presented as an extra environmental character set to one if the energy level is above 0.5, otherwise zero.

While the optimum steps to goal remains 1.7 as in the other parallel multi-objective tasks in the Woods1e environment, the optimum average reward is no longer 1000 due to the dynamic reward. If the energy level at the goal state is above 0.5, achieving the correct ‘food’ goal will gain a reward in the range [1000, 500]. The same is true when the animat correctly reaches the ‘energy’ goal with its energy below 0.5. Assuming an equal random distribution of initial energy levels for trials, the average reward for success is 750.

Figure 4-22 and Figure 4-23 show results using random exploration, while Figure 4-24 and Figure 4-25 show the same population sizes with roulette-wheel exploration. Using both methods of exploration, XCS was able to solve this problem with a minimum of adjustment, achieving simultaneously optimal performance both in terms of reward achieved and steps taken.

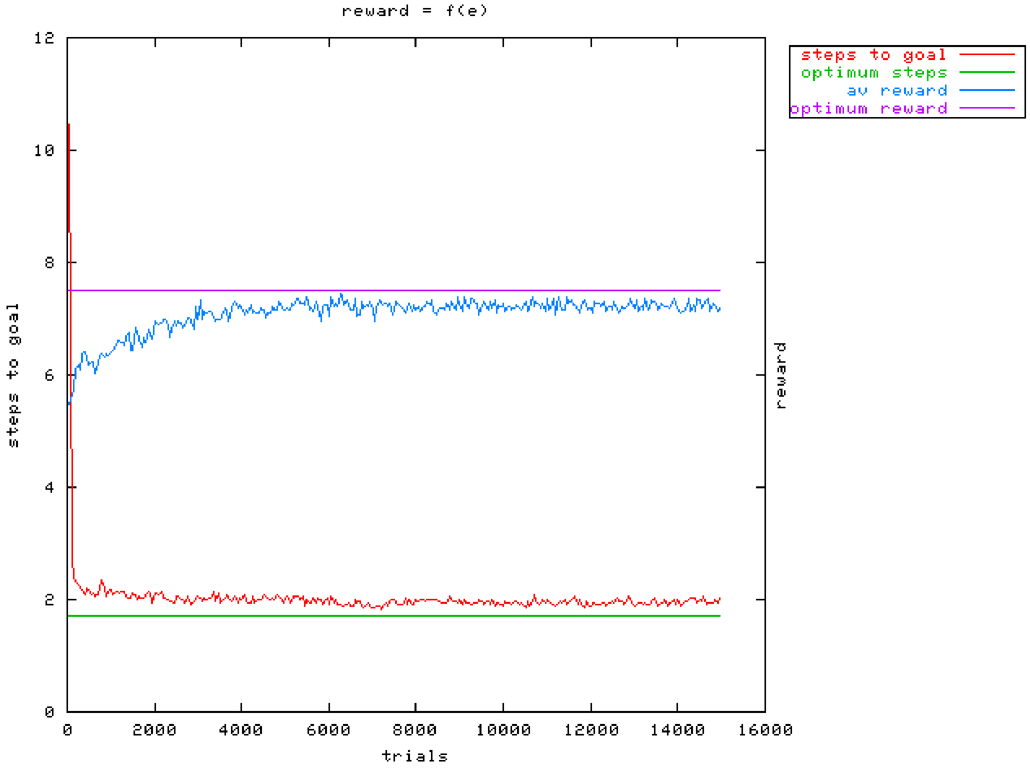


Figure 4-22 Woods1e-type3. *N*=1600, random exploration

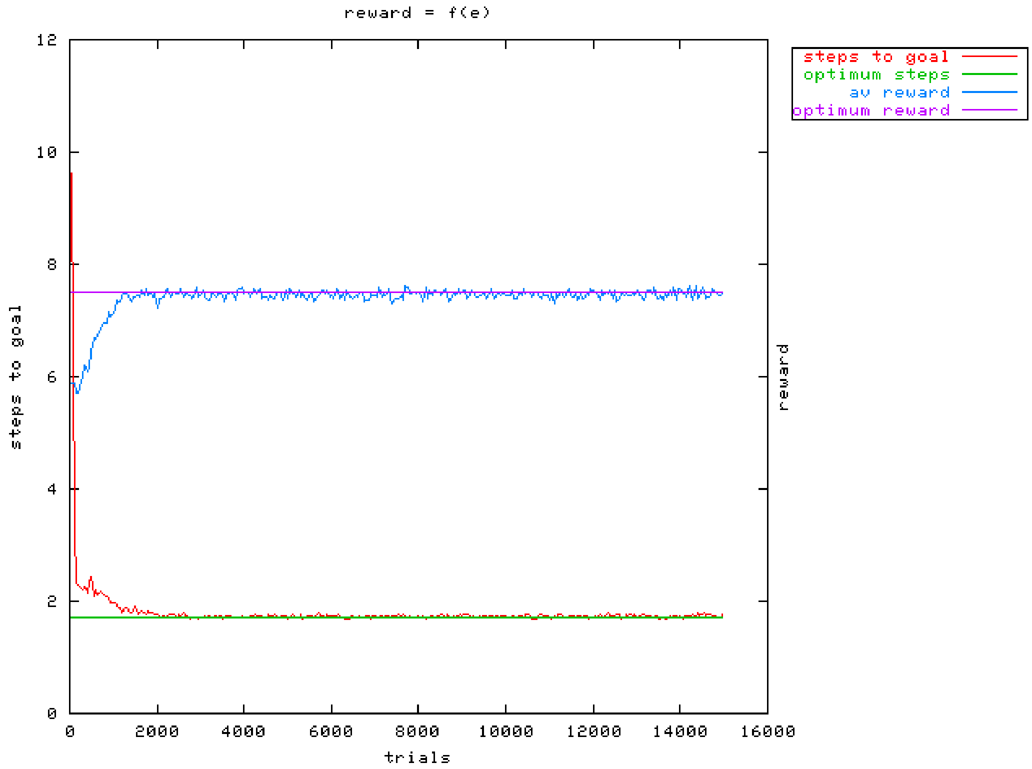


Figure 4-23 Woods1e-type3. *N*=3200, random exploration

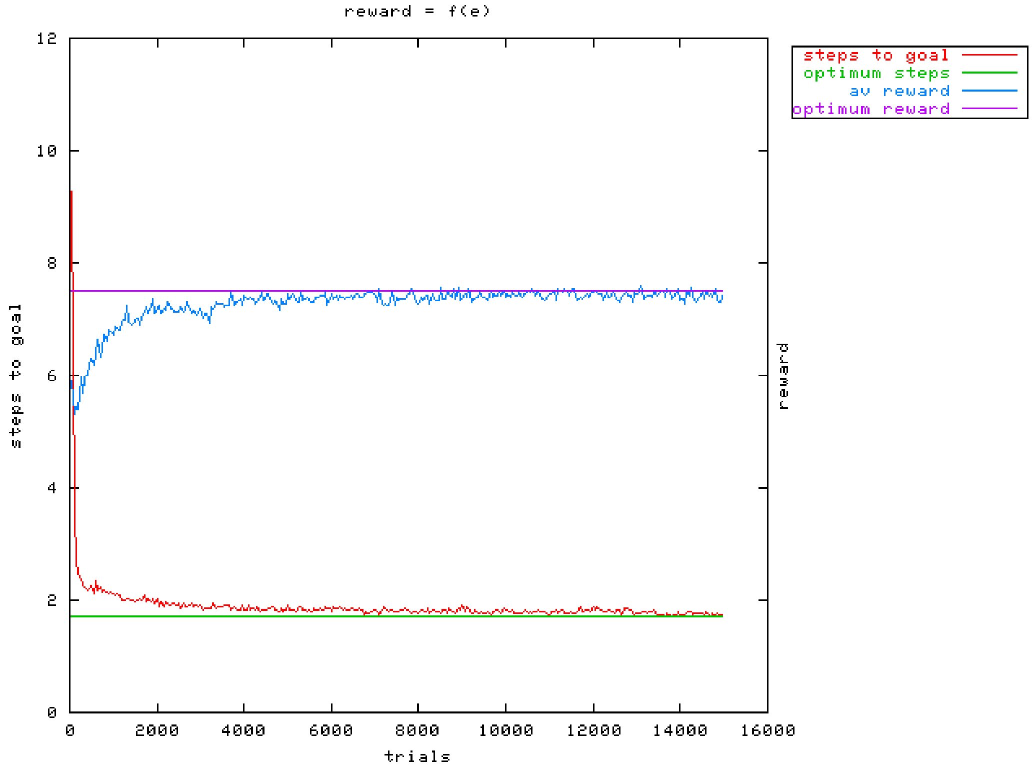


Figure 4-24 Woods1e-type3. *N*=1600, roulette exploration

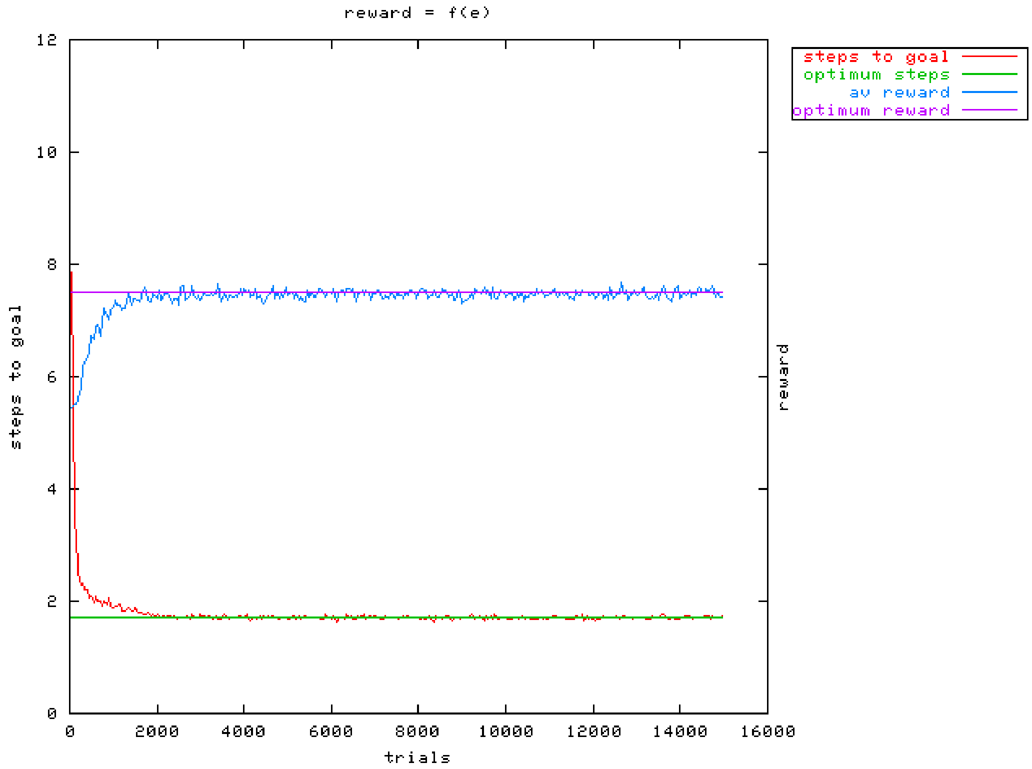


Figure 4-25 Woods1e-type3. *N*=3200, roulette exploration

We have so far observed that XCS is generally less able to achieve such good results with smaller populations when using roulette wheel exploration, compared with random exploration. As can be seen from Figs 5-19 – 5-22, this is not the case with this problem. Interestingly, with *N*=1600, the treatment using roulette exploration out-performs the equivalent treatment using random exploration both in terms of reducing to an optimum the average steps taken to the goal state (roulette 1.76 ±0.03, random 1.98 ±0.04), and also with respect to the average reward gained (roulette 743 ±7.34, random 714 ±9.97). This result is returned to in section 4.7.

	t Stat	t Critical two-tail
Steps to Goal	1.850	2.131
Reward	0.780	2.101

Table 4-4 Two Tailed Student t-test, Roulette & Random N=3200.

No significant difference at 95% confidence level.

4.5 Increasing Complexity

4.5.1 More objectives

The complexity of the Woods1e task is again increased by adding a third goal state. The new goal, 'maintenance', takes priority over the other two goals. The '*Woods1em*' environment is show in Figure 4-26.

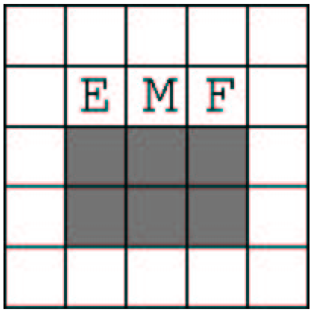


Figure 4-26 The 'Woods1em' 3 objective environment

The reward function changes slightly to accommodate the third objective, as represented in Figure 4-27.

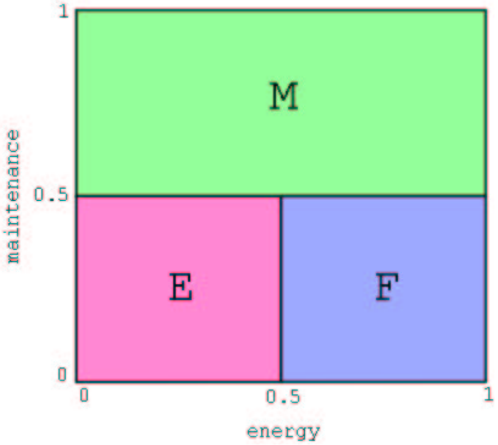


Figure 4-27 Graphical representation of correct goal in 3 objective Woods problem

If the animat reaches the 'maintenance' goal and its need for maintenance is higher than or equal to 0.5, it is rewarded 1000, otherwise 1, irrespective of the internal energy level. If the animat reaches the ‘food’ goal when the need for maintenance is lower than 0.5 and the energy level is higher than 0.5 it receives a high reward (1000), otherwise low (1). Conversely, if the animat reaches the ‘energy’ goal when the need for maintenance is lower than 0.5 and the energy level is lower than or equal to 0.5 it receives a high reward (1000), otherwise low (1).

All other experimental details remain as before, except that the condition part of the classifier is extended by a further character which is set to zero if the animat’s maintenance level is lower than 0.5, and otherwise set to one. As is the case with the internal energy level, the animat’s ‘need’ for maintenance is set randomly between 0 and 1 at the start of each trial, such that all combinations of high and low maintenance need, and high and low energy need are represented in equal numbers of trials. The optimum steps to goal is 1.8, and the optimum average reward that can be achieved is once again 1000.

4.5.1.1.1 Stepwise reward, no cost of movement(Woods1em-type1)

As shown in Figure 4-28, XCS easily achieves an optimal average reward and steps to goal using *N*=3200 and random exploration. With an equivalent population size (Figure 4-29), roulette-wheel exploration takes longer to achieve similar performance, and the system shows more variability in the average reward achieved. Increasing the population size allowed XCS with roulette-wheel exploration to achieve results that match those with random exploration, as shown in Figure 4-30.

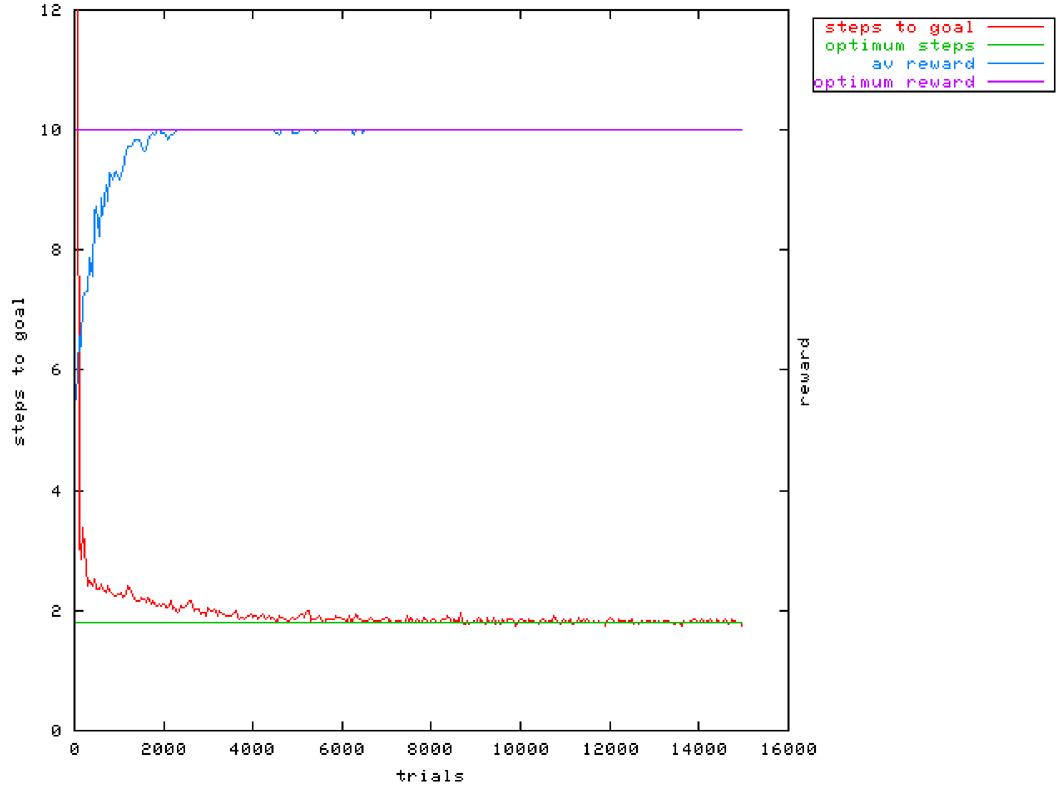


Figure 4-28 Woods1em-type1. *N*=3200, random exploration.

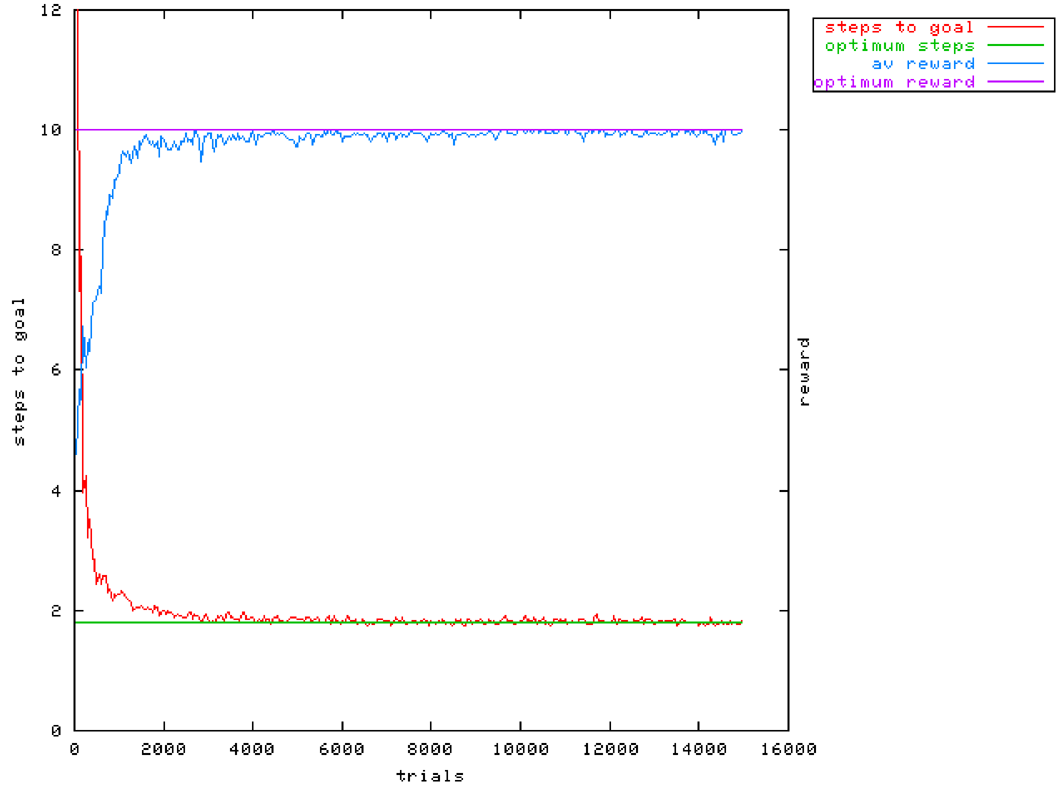


Figure 4-29 Woods1em-type1. *N*=3200, roulette exploration.

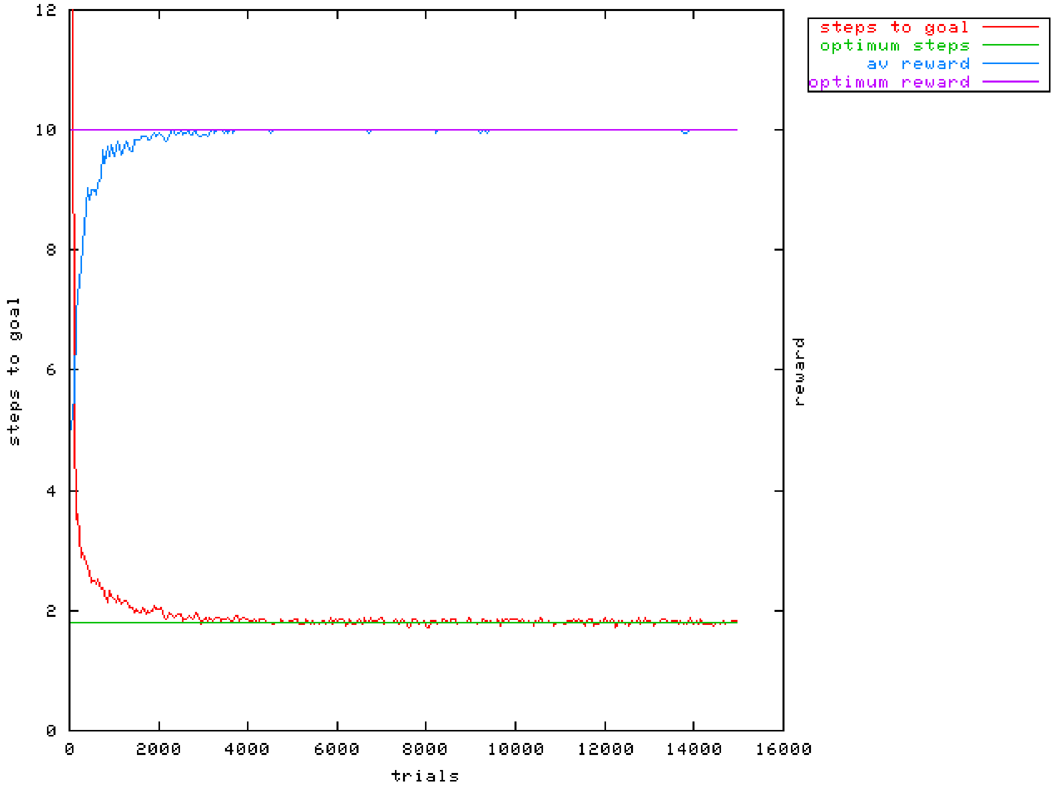


Figure 4-30 Woods1em-type1. *N*=6400, roulette exploration.

	t Stat	t Critical two-tail
Steps to Goal	0.240	2.131
Reward	-	-

Table 4-5 Two Tailed Student t-test, Roulette N=6400, Random N=3200

As can be seen from Table 4-5, there is no significant difference in steps to goal at *p*=0.05. Figures for reward are not included as both treatments were identically perfect with no variance.

4.5.1.1.2 Stepwise reward, dynamic cost (Woods1em-type2)

Having shown that XCS is easily capable of optimal performance in the three objective Woods1em environment with no change in the parameters that were used in the simpler two objective environment, once again a cost of movement of 0.01 ‘energy points’ is introduced for each move taken by the animat.

Figure 4-31 and Figure 4-32 show that, although XCS was unable to achieve optimal performance with *N*=1600 and random exploration, increasing *N* to 3200 allowed optimal performance.

Figure 4-33, Figure 4-34 and Table 4-6 show that equivalent performance can be achieved using roulette-wheel exploration, but only by increasing *N* once more.

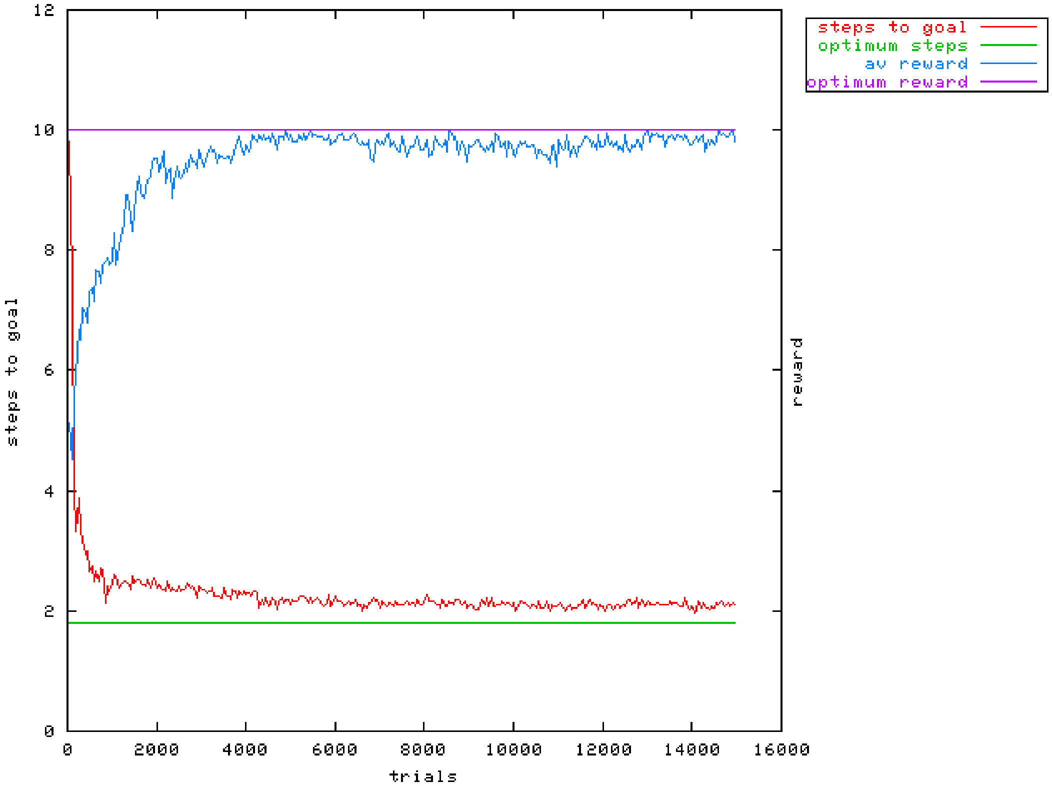


Figure 4-31 Woods1em-type2. *N*=1600, random exploration.

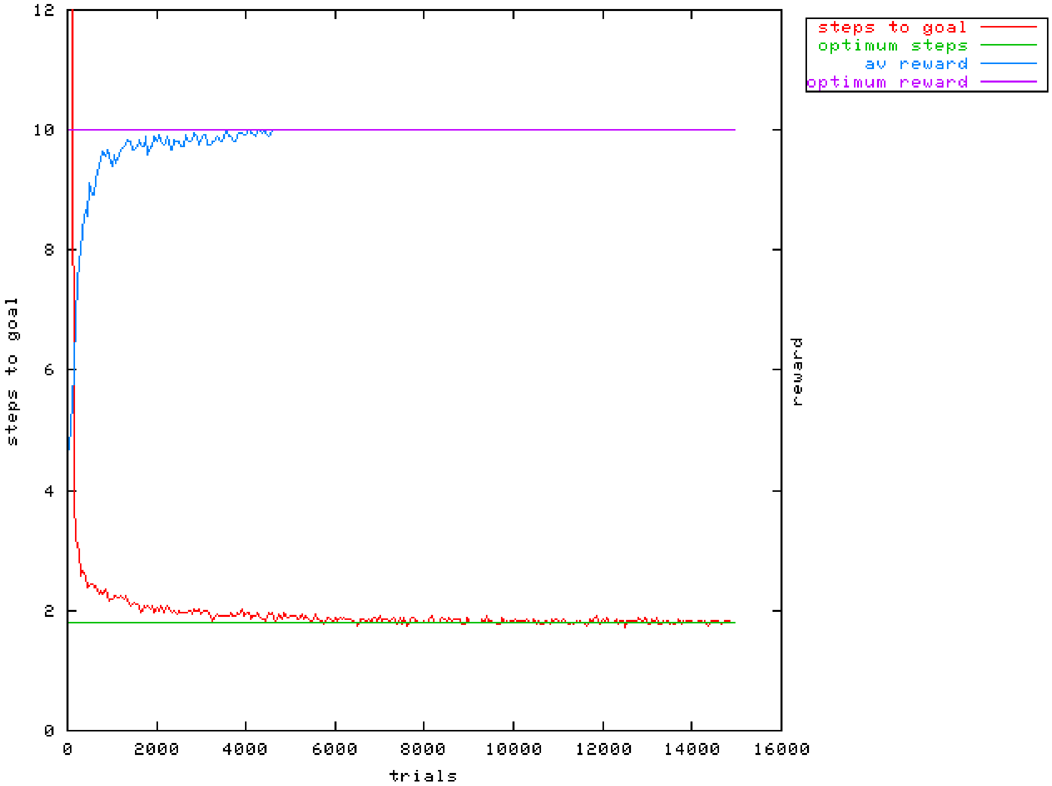


Figure 4-32 Woods1em-type2. ***N*=3200, random exploration.**

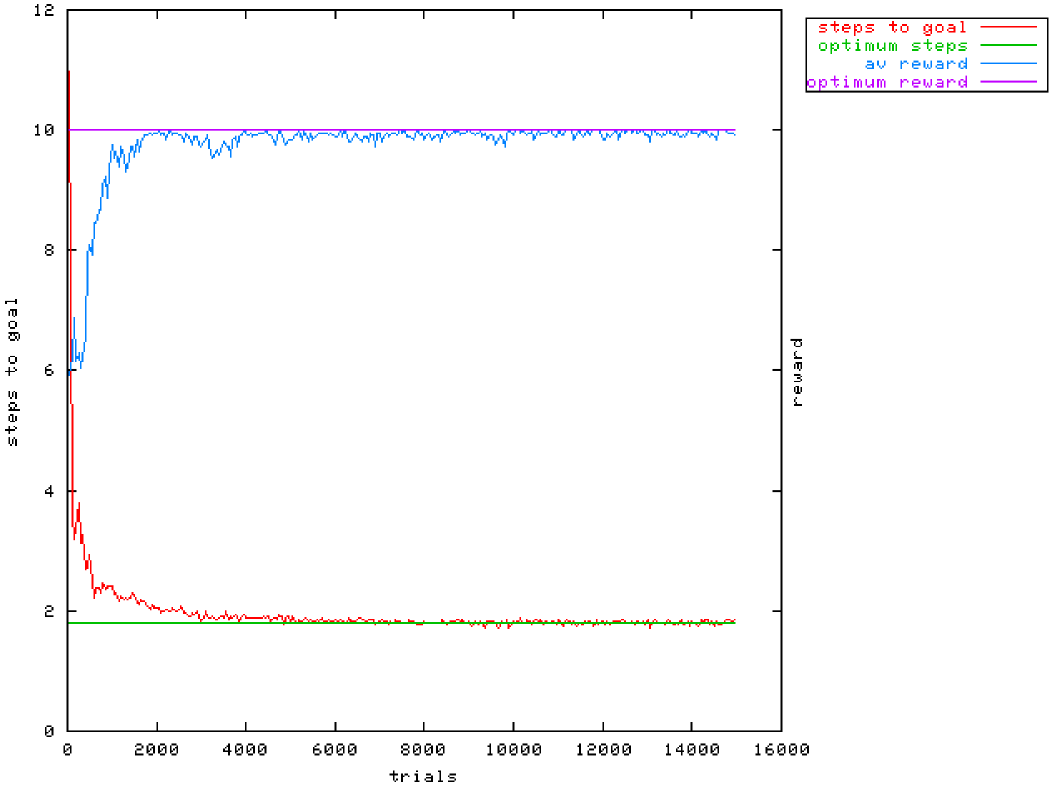


Figure 4-33 Woods1em-type2. ***N*=3200, roulette exploration.**

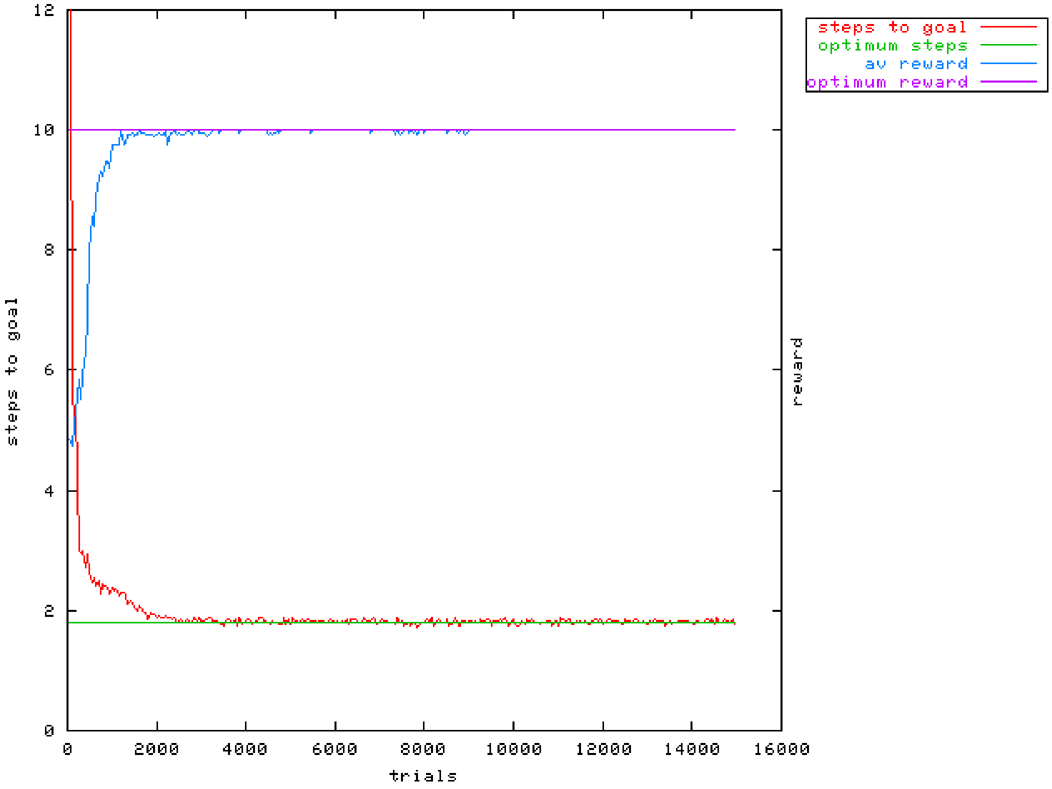


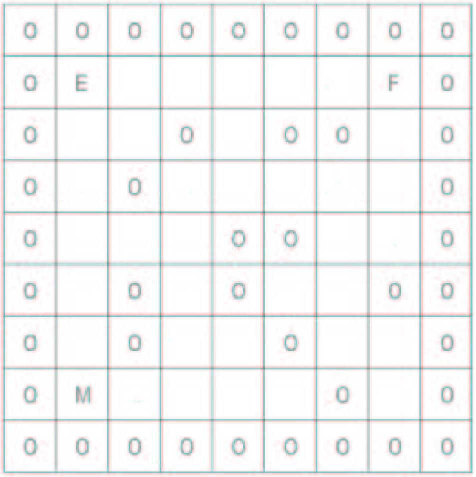
Figure 4-34 Woods1em-type2. *N*=6400, roulette exploration.

	t Stat	t Critical two-tail
Steps to Goal	0.303	2.145
Reward	-	-

Table 4-6 Two Tailed Student t-test, Roulette N=6400, Random N=3200. No significant difference at 95% confidence level.

4.5.2 Bigger Environment

The complexity of the task is now further increased by enlarging the environment. An adapted version of the well-known Maze5 grid-world is used [Lanzi1999], termed Maze5em. Figure 4-35 shows Maze5em – a bounded environment containing obstacles (‘O’) and three goal states; Food (‘F’), Energy, (‘E’) and Maintenance (‘M’). As before, all environmental states are encoded in three characters, and eight actions are possible.



O	O	O	O	O	O	O	O	O
O	E						F	O
O			O		O	O		O
O		O						O
O				O	O			O
O		O		O			O	O
O		O			O			O
O	M					O		O
O	O	O	O	O	O	O	O	O

Figure 4-35 The Maze5em 3-objective grid-world

The payoff matrix remains the same as for the three objective Woods1em problem, as outlined above (Figure 4-27).

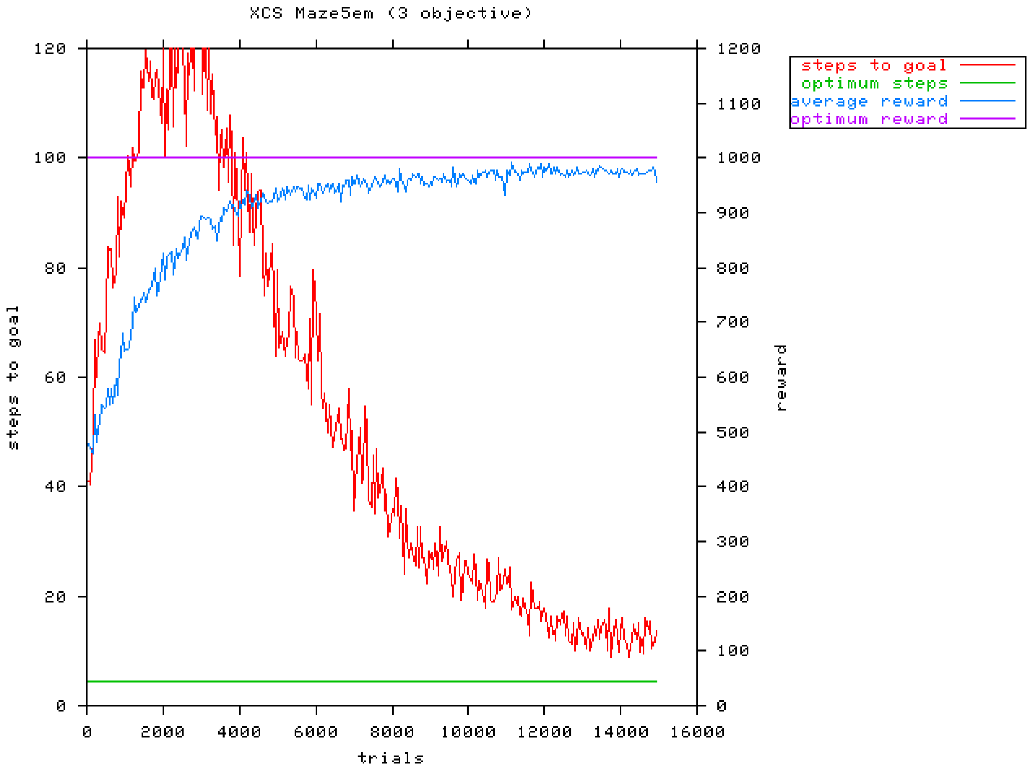


Figure 4-36 Maze5em, random exploration, *N*=6400, *p#* =0.33

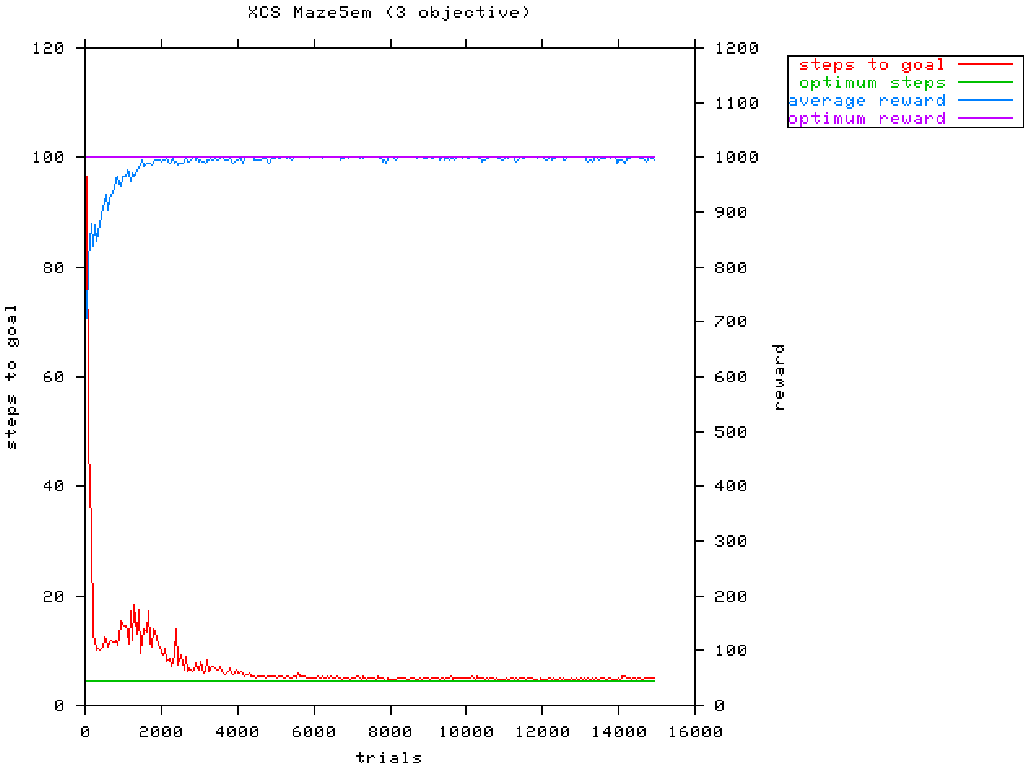


Figure 4-37 Maze5em, random exploration, *N*=6400, *p#*=0.1

Figure 4-36 and Figure 4-37 show the performance of XCS with *N*=6400 and *p#*= 0.33, and 0.1 respectively. XCS achieve both optimal steps to goal (4.8) and an optimal average reward (1000) in the Maze5em environment using a random action selection policy in the 50% of trials that are intended for 'exploration' when *p#* = 0.1. When *p#*=0.33, some runs are optimal, but some are not.

However, it was less easy to achieve similar performance using a roulette-wheel exploration policy. Once again, it was found that increasing the value of *N* was helpful, as was reducing the value of p#, as shown in Figure 4-38 - Figure 4-40. (Table 4-7 shows that the results gained with roulette wheel selection at *N*=8000 and *p#*=0.1 are significantly different from those gained with random selection at *N*=6400, *p#*=0.1. However, the improvements with increasing *N* and decreasing *p#* are obvious.)

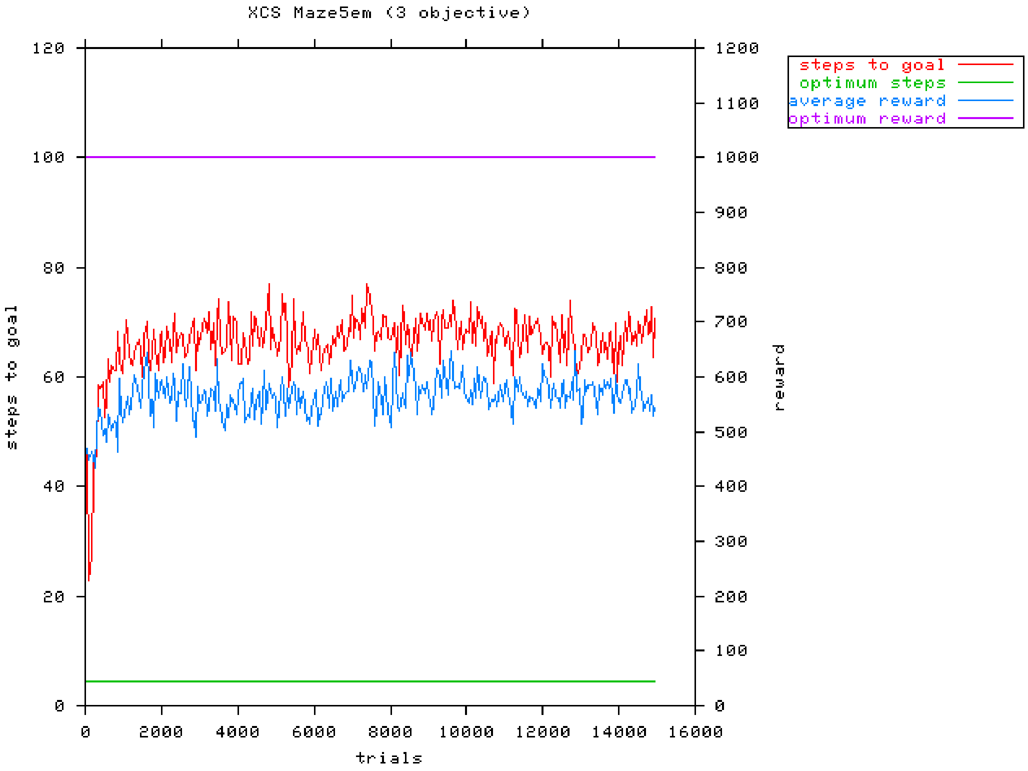


Figure 4-38 Maze5em, roulette exploration, *N*=6400, p#=0.33

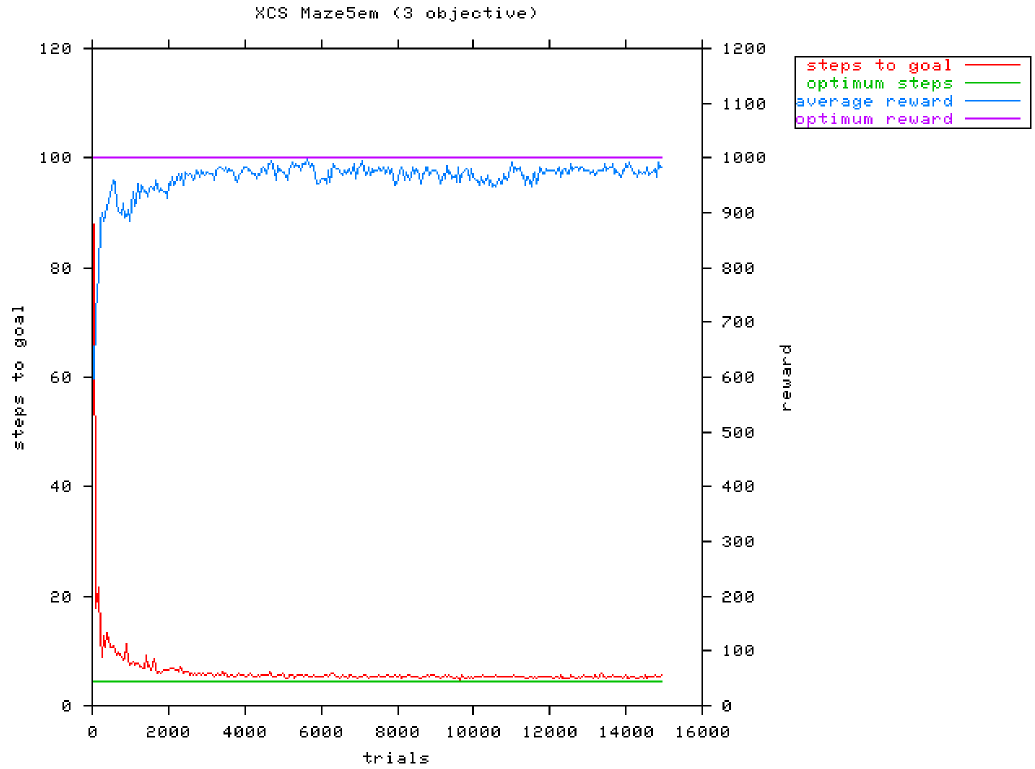


Figure 4-39 Maze5em, roulette exploration, *N*=6400, p#=0.1

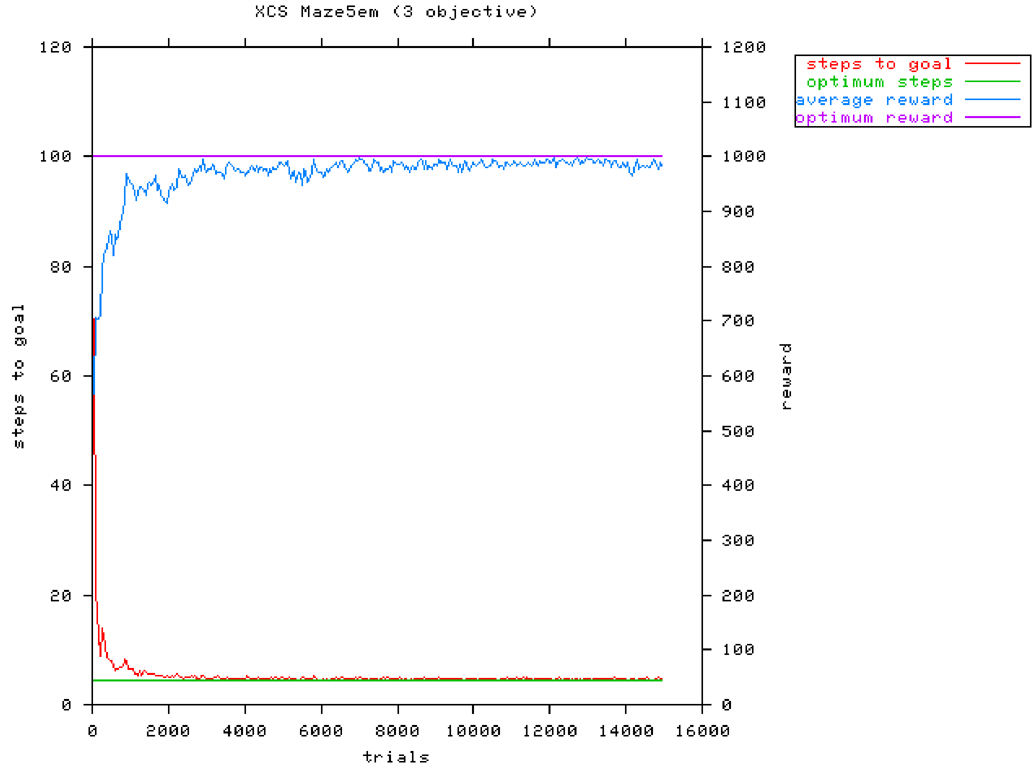


Figure 4-40 Maze5em, roulette exploration, *N*=8000, p#=0.1

	t Stat	t Critical two-tail
Steps to Goal	4.304	2.110
Reward	4.304	2.110

Table 4-7 Two Tailed Student t-test, Roulette N=8000, Random N=6400, *p#*=0.1

This behaviour is interesting. In the majority of cases, we have found that roulette-wheel exploration can produce comparable results to that achieved using a random action selection policy at the expense of increasing *N*. We now see that in this latter case it is not enough to increase *N*, we also need to decrease p#.

P# is the probability that an allele in the condition of a classifier will have the 'don't care' symbol, '#'. It would therefore appear that XCS is suffering from a tendency to produce over-general classifiers in Maze5em. This tendency may be exacerbated when using the roulette-wheel action selection policy in the explore phase.

Lanzi makes the following observations in his investigations of generalization in XCS [Lanzi1999].

- In the Maze5 environment with *N*=1600, Lanzi reports that 'biased exploration' gives better results than standard XCS. 'Biased exploration chooses an action randomly with a probability P_s=0.3'. In the normal 50:50 balance between random exploration and deterministic exploitation, this value would be 0.5. Biased exploration therefore results in less exploration.
- Over-generalization stops XCS from reaching optimal performance. When *p#*=0, XCS is an approximation of a 'tabular Q-learner' and achieves optimality in Maze5. Decreasing *p#* may help XCS in problems where there are fewer generalizations - this is the 'schema challenge' [Butz2003b] in operation.
- Lanzi suggests that over-generals arise because some states are visited less often than others, so classifiers which are accurate in a frequently-visited niche but less accurate

in a niche which is less often visited – for example, one further from the goal – have a high fitness and are more likely to be reproduced. The GA chooses parents from [A], so there will be fewer chances to discover optimal rules in infrequently visited states. He introduces a new operator '*tele-transportation*', which restarts an exploration trial at a new randomly selected position if the animat takes more than a maximum number of steps before reaching the goal. This in effect addresses the issues of the 'class imbalance problem'. It is known that datasets in which one class is represented by more instances than other classes present difficulties to machine learning techniques [Japkowicz2002]. Techniques that attempt to discover generalizations are likely to be more susceptible to this problem than those that do not, such as k-Nearest Neighbour. Recent work on an accuracy-based classifier system, UCS, shows that the class-imbalance problem affects LCSs [Orriols2005a, Orriols2005b]. In essence, Lanzi's *teletransportation* implements a naïve form of oversampling.

- In summary, Lanzi states; 'XCS fails to learn an optimal policy in environments where the system is not very likely to explore all the environmental niches frequently'. This is contradicted by the results he presents using biased exploration.

We have observed that lowering p# improves XCS' performance in the three objective Maze5em problem in the same way it does in Maze5. If roulette-wheel selection is in effect 'annealing too fast', then there will be insufficient exploration. There may be greater variation between experiments; in some, the best actions in any particular environmental state are not explored sufficiently and over-general classifiers are allowed to direct the system into sub-optimal actions.

4.6 Roulette-wheel Exploration - Discussion

The hypothesis next tested is that increasing *N* with roulette selection improves performance by decreasing the tendency towards excessive generalization. For each experimental treatment of increasing values of *N*, with p#=0.33 and roulette-wheel exploration, the average specificity is shown for all classifiers which match in each environmental state. These figures are the end result after 15000 exploit trials, and are the average of ten runs.

As can be seen from Table 4-8, with increasing values of *N* there is a tendency for the amount of generalization to drop. Shaded values indicate that optimal performance was not achieved. Considered over all states in the environment, the average percentage specificity (i.e., the average percentage of non-‘#’ alleles in the classifiers’ conditions) is;

	Random Exploration		*Roulette Exploration*	
N	p# 0.33	p# 0.1	p# 0.33	p# 0.1
1600	82	80	10	75
6400	87	87	18	88
8000		90		90

Table 4-8 Percentage specificity in all matching classifiers

- We have seen that as *N* increases, performance improves.
- It is interesting to note that although the two treatments that used random exploration at *N*=6400 had the same overall specificity, it was only by using a low value of *p#* that consistently optimal performance was produced.
- There is a general tendency for specificity to increase with *N*.
- There is a dramatic difference between the results with roulette selection and *p#* = 0.33, and those of all other treatments.

It seems safe to assume that increasing *N* reduces the speed at which the roulette-wheel action selection anneals. A larger population size allows more diversity to be maintained, giving the GA more time to weed out the over-general rules.

In [Butz2003a], the *reproductive opportunity bound* (ROP-bound) upon *N* is derived. Given that there are no effects of action set size or fitness influences in the deletion process, they show that,

$$N > n\,2^{kd+(l-kd)s[p]+1}$$

where *S[p]* is the value to which the specificity of the population would converge if no fitness influence was present, and *n* is the number of actions. k_d is the ‘difficulty’ of the problem – determined by the length of the minimum order schema that provides guidance on whether one solution is better than another (they illustrate this with the extreme case of the *parity problem* in which all bits must be specified to correctly predict the outcome; in such a case any partial solution is as little use as a completely general classifier).

It can be seen from the above equation that *N* must grow with bigger values of both the order of difficulty k_d, and also the necessary specificity *S[p]* in the population. This is based upon the assumption that ‘binary input strings are encountered which are uniformly distributed over the whole problem instance space $\{0, 1\}^l$.’

The Maze5 environment is one in which little generalization is possible, and thus is difficult for XCS. To achieve optimal performance with either action selection mechanism required more specificity (i.e. a lower value of *p#*). This is as predicted by the ROP-bound, above. The assumption that ‘binary input strings are encountered ... uniformly…’ will be true neither in multi-step problems (states further from the goal are less frequently visited), nor when the action selection scheme is other than random. The necessity to increase *N* when using roulette wheel exploration must be due to the effects of bias in sampling frequency on the ROP-bound. In further support of the predictions of Butz et al, we would expect that given a sufficient value of *N* and *S[p]* to satisfy the ROP-bound, and for a given value of μ (assumed to be sub-optimal), as we further increase *N* we will achieve optimal performance sooner. This is indeed the case, as is shown in Figure 4-41 and Figure 4-42.

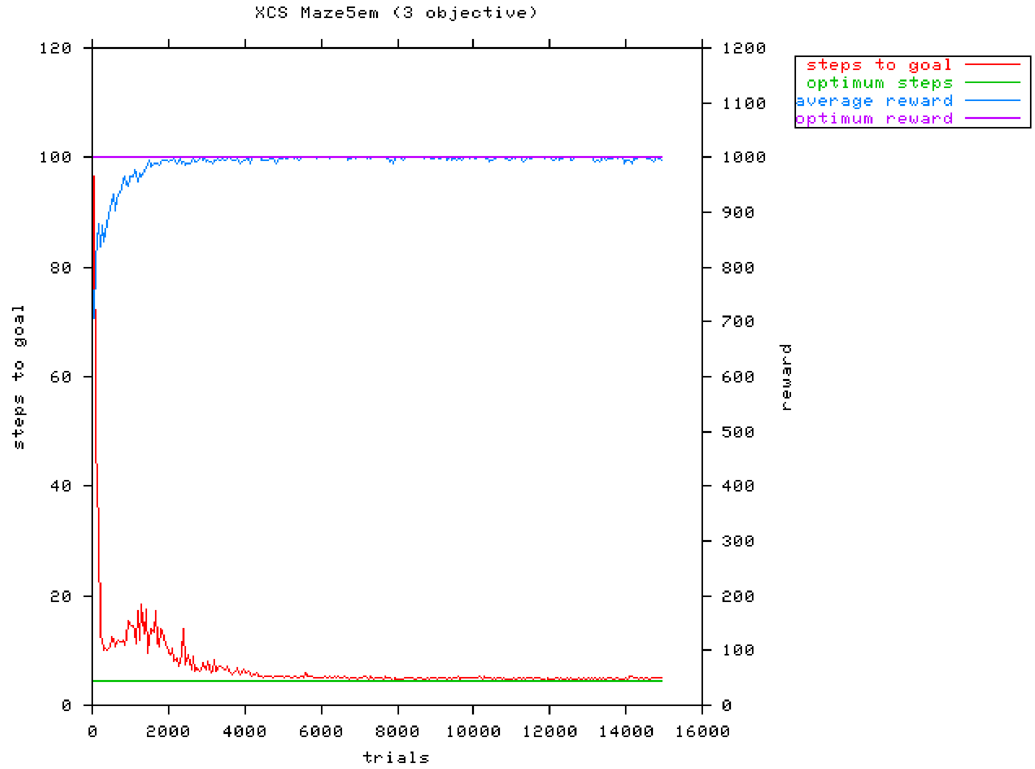


Figure 4-41 ***N*=6400, *p#*=0.1 Optimal performance achieved after approx 4500 trials.**

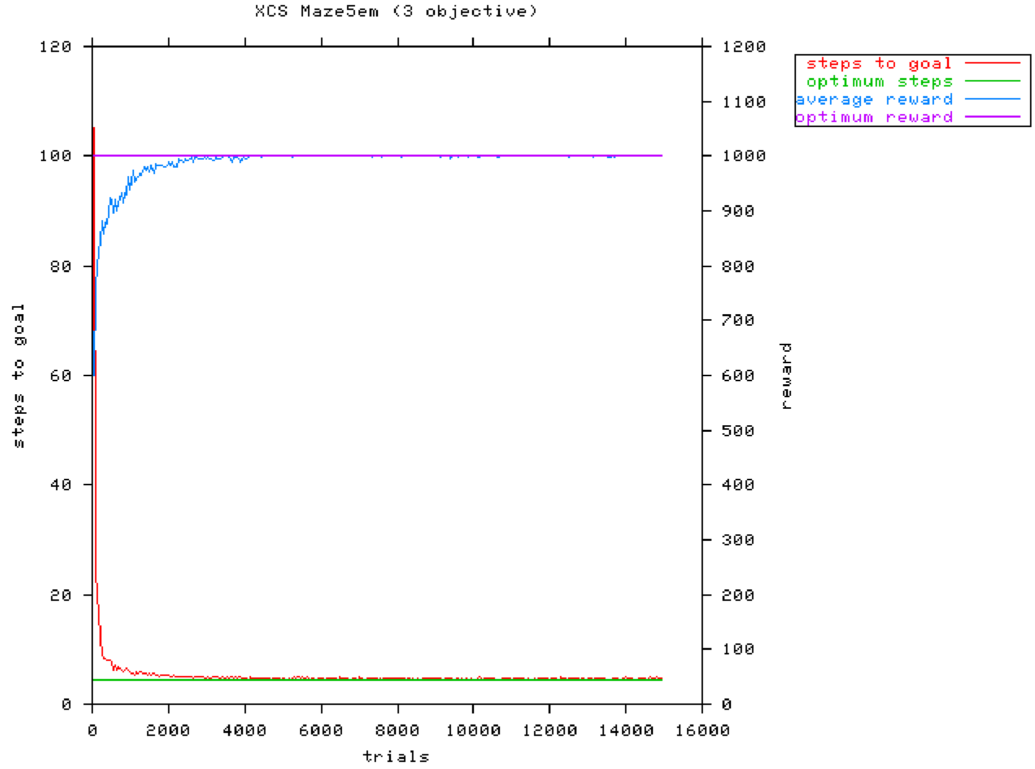
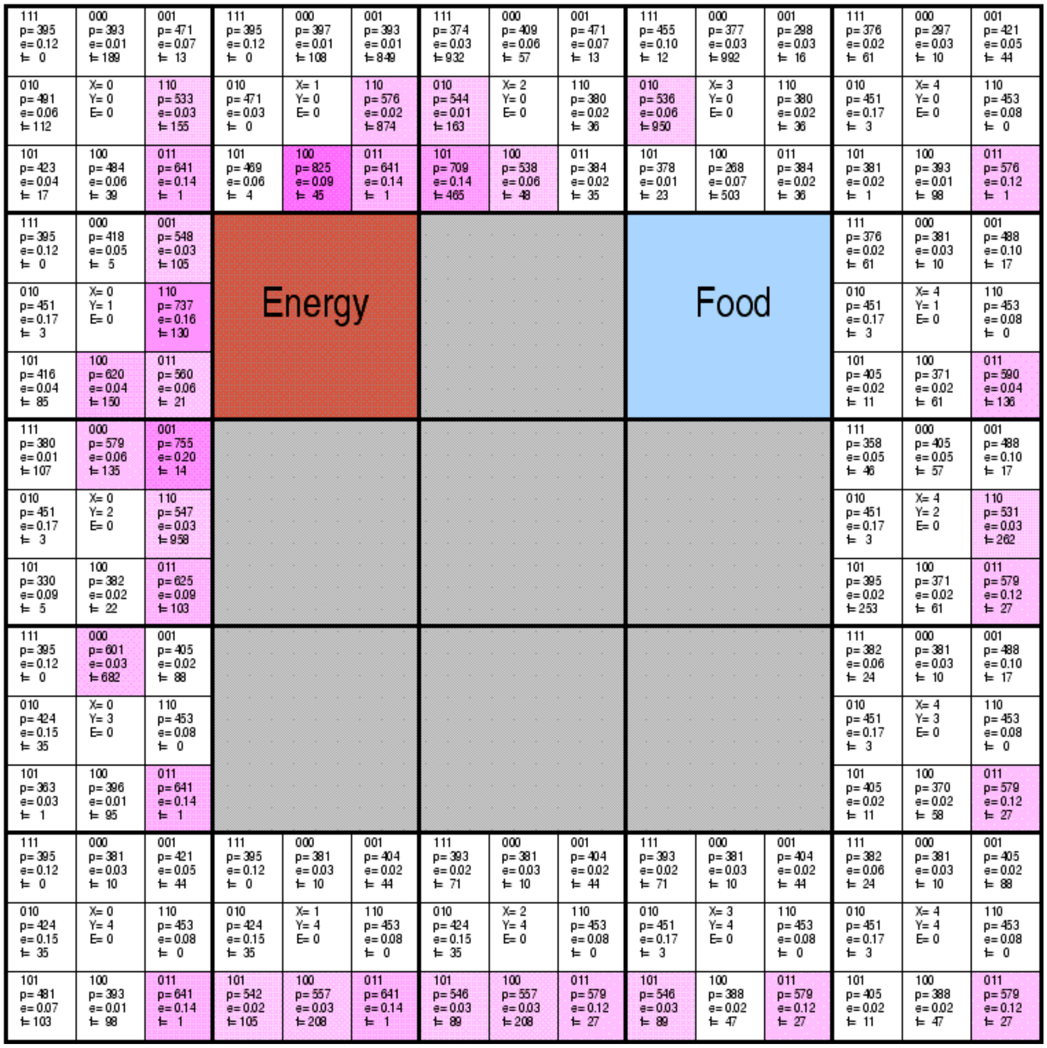


Figure 4-42 ***N*=8000, *p#*=0.1 Optimal performance achieved by approx 2500 trials.**

Generally then, we have seen that random exploration gives better results for a given value of *N* than roulette wheel exploration. However, it will be recalled that when we examined the behaviour of XCS using random and roulette action selection on the Woods1e problem with a continuous reward function and associated cost of movement (Woods1e-type3), we noticed that with lower values of *N*, roulette selection outperformed random selection. This contradicts all other results, and requires explanation.

Figure 4-43 and Figure 4-44 show a map of all state-action pairs in the Woods1e environment. For each state-action pair the action encoding is shown, with the values of prediction (*p*), error/1000 (*e*), and fitness*1000 (*f*) for the classifier in [M] with the highest *P* value for this action – the highest of these then indicates which action would be taken in the deterministic phase. Figure 4-43 shows values when the internal energy level is below 0.5; i.e. the learner should seek the energy goal. Figure 4-44 shows the opposite case, the ‘food goal’. In both figures, state-action pairs for which the representative classifier has a *p* value over 500 are shaded proportionally to the *p* value. A state-action cell with a *p*-value over 500 indicates ‘a step in the right direction’. The darker the shading, the more likely an action will be chosen by roulette selection.

It will be seen that there are more shaded boxes in Figure 4-43 than in Figure 4-44. This indicates that using random selection with *N*=1600, XCS has concentrated its resources on the ‘Energy Trials’ where *e*<0.5, in which it seems to point towards the shortest path to goal from each state space. In the ‘Food Trials’, few values are over 500, indicating that there is little knowledge of where the best reward may be found. Using random exploration with *N*=1600, XCS has nearly solved ‘half the problem’.



111 p= 395 e= 0.12 t= 0	000 p= 393 e= 0.01 t= 189	001 p= 471 e= 0.07 t= 13	111 p= 395 e= 0.12 t= 0	000 p= 397 e= 0.01 t= 108	001 p= 393 e= 0.01 t= 849	111 p= 374 e= 0.03 t= 932	000 p= 409 e= 0.06 t= 57	001 p= 471 e= 0.07 t= 13	111 p= 455 e= 0.10 t= 12	000 p= 377 e= 0.03 t= 992	001 p= 298 e= 0.03 t= 16	111 p= 376 e= 0.02 t= 61	000 p= 297 e= 0.03 t= 10	001 p= 421 e= 0.05 t= 44
010 p= 491 e= 0.06 t= 112	X= 0 Y= 0 E= 0	110 p= 533 e= 0.03 t= 155	010 p= 471 e= 0.03 t= 0	X= 1 Y= 0 E= 0	110 p= 576 e= 0.02 t= 874	010 p= 544 e= 0.01 t= 163	X= 2 Y= 0 E= 0	110 p= 380 e= 0.02 t= 36	010 p= 536 e= 0.06 t= 950	X= 3 Y= 0 E= 0	110 p= 380 e= 0.02 t= 36	010 p= 451 e= 0.17 t= 3	X= 4 Y= 0 E= 0	110 p= 453 e= 0.08 t= 0
101 p= 423 e= 0.04 t= 17	100 p= 484 e= 0.06 t= 39	011 p= 641 e= 0.14 t= 1	101 p= 469 e= 0.06 t= 4	100 p= 825 e= 0.09 t= 45	011 p= 641 e= 0.14 t= 1	101 p= 709 e= 0.14 t= 465	100 p= 538 e= 0.06 t= 48	011 p= 384 e= 0.02 t= 35	101 p= 378 e= 0.01 t= 23	100 p= 268 e= 0.07 t= 503	011 p= 384 e= 0.02 t= 36	101 p= 381 e= 0.02 t= 1	100 p= 393 e= 0.01 t= 98	011 p= 576 e= 0.12 t= 1
111 p= 395 e= 0.12 t= 0	000 p= 418 e= 0.05 t= 5	001 p= 548 e= 0.03 t= 105	Energy						Food			111 p= 376 e= 0.02 t= 61	000 p= 381 e= 0.03 t= 10	001 p= 488 e= 0.10 t= 17
010 p= 451 e= 0.17 t= 3	X= 0 Y= 1 E= 0	110 p= 737 e= 0.16 t= 130										010 p= 451 e= 0.17 t= 3	X= 4 Y= 1 E= 0	110 p= 453 e= 0.08 t= 0
101 p= 416 e= 0.04 t= 85	100 p= 620 e= 0.04 t= 150	011 p= 560 e= 0.06 t= 21										101 p= 405 e= 0.02 t= 11	100 p= 371 e= 0.02 t= 61	011 p= 590 e= 0.04 t= 136
111 p= 380 e= 0.01 t= 107	000 p= 579 e= 0.06 t= 135	001 p= 755 e= 0.20 t= 14										111 p= 358 e= 0.05 t= 46	000 p= 405 e= 0.05 t= 57	001 p= 488 e= 0.10 t= 17
010 p= 451 e= 0.17 t= 3	X= 0 Y= 2 E= 0	110 p= 547 e= 0.03 t= 958										010 p= 451 e= 0.17 t= 3	X= 4 Y= 2 E= 0	110 p= 531 e= 0.03 t= 262
101 p= 330 e= 0.09 t= 5	100 p= 382 e= 0.02 t= 22	011 p= 625 e= 0.09 t= 103										101 p= 395 e= 0.02 t= 253	100 p= 371 e= 0.02 t= 61	011 p= 579 e= 0.12 t= 27
111 p= 395 e= 0.12 t= 0	000 p= 601 e= 0.03 t= 682	001 p= 405 e= 0.02 t= 88										111 p= 382 e= 0.06 t= 24	000 p= 381 e= 0.03 t= 10	001 p= 488 e= 0.10 t= 17
010 p= 424 e= 0.15 t= 35	X= 0 Y= 3 E= 0	110 p= 453 e= 0.08 t= 0										010 p= 451 e= 0.17 t= 3	X= 4 Y= 3 E= 0	110 p= 453 e= 0.08 t= 0
101 p= 363 e= 0.03 t= 1	100 p= 396 e= 0.01 t= 95	011 p= 641 e= 0.14 t= 1										101 p= 405 e= 0.02 t= 11	100 p= 370 e= 0.02 t= 58	011 p= 579 e= 0.12 t= 27
111 p= 395 e= 0.12 t= 0	000 p= 381 e= 0.03 t= 10	001 p= 421 e= 0.05 t= 44	111 p= 395 e= 0.12 t= 0	000 p= 381 e= 0.03 t= 10	001 p= 404 e= 0.02 t= 44	111 p= 393 e= 0.02 t= 71	000 p= 381 e= 0.03 t= 10	001 p= 404 e= 0.02 t= 44	111 p= 393 e= 0.02 t= 71	000 p= 381 e= 0.03 t= 10	001 p= 404 e= 0.02 t= 44	111 p= 382 e= 0.06 t= 24	000 p= 381 e= 0.03 t= 10	001 p= 405 e= 0.02 t= 88
010 p= 424 e= 0.15 t= 35	X= 0 Y= 4 E= 0	110 p= 453 e= 0.08 t= 0	010 p= 424 e= 0.15 t= 35	X= 1 Y= 4 E= 0	110 p= 453 e= 0.08 t= 0	010 p= 424 e= 0.15 t= 35	X= 2 Y= 4 E= 0	110 p= 453 e= 0.08 t= 0	010 p= 451 e= 0.17 t= 3	X= 3 Y= 4 E= 0	110 p= 453 e= 0.08 t= 0	010 p= 451 e= 0.17 t= 3	X= 4 Y= 4 E= 0	110 p= 453 e= 0.08 t= 0
101 p= 481 e= 0.07 t= 103	100 p= 393 e= 0.01 t= 98	011 p= 641 e= 0.14 t= 1	101 p= 542 e= 0.02 t= 105	100 p= 557 e= 0.03 t= 208	011 p= 641 e= 0.14 t= 1	101 p= 546 e= 0.03 t= 89	100 p= 557 e= 0.03 t= 208	011 p= 579 e= 0.12 t= 27	101 p= 546 e= 0.03 t= 89	100 p= 388 e= 0.02 t= 47	011 p= 579 e= 0.12 t= 27	101 p= 405 e= 0.02 t= 11	100 p= 388 e= 0.02 t= 47	011 p= 579 e= 0.12 t= 27

Figure 4-43 Random exploration. 'Energy Trial'

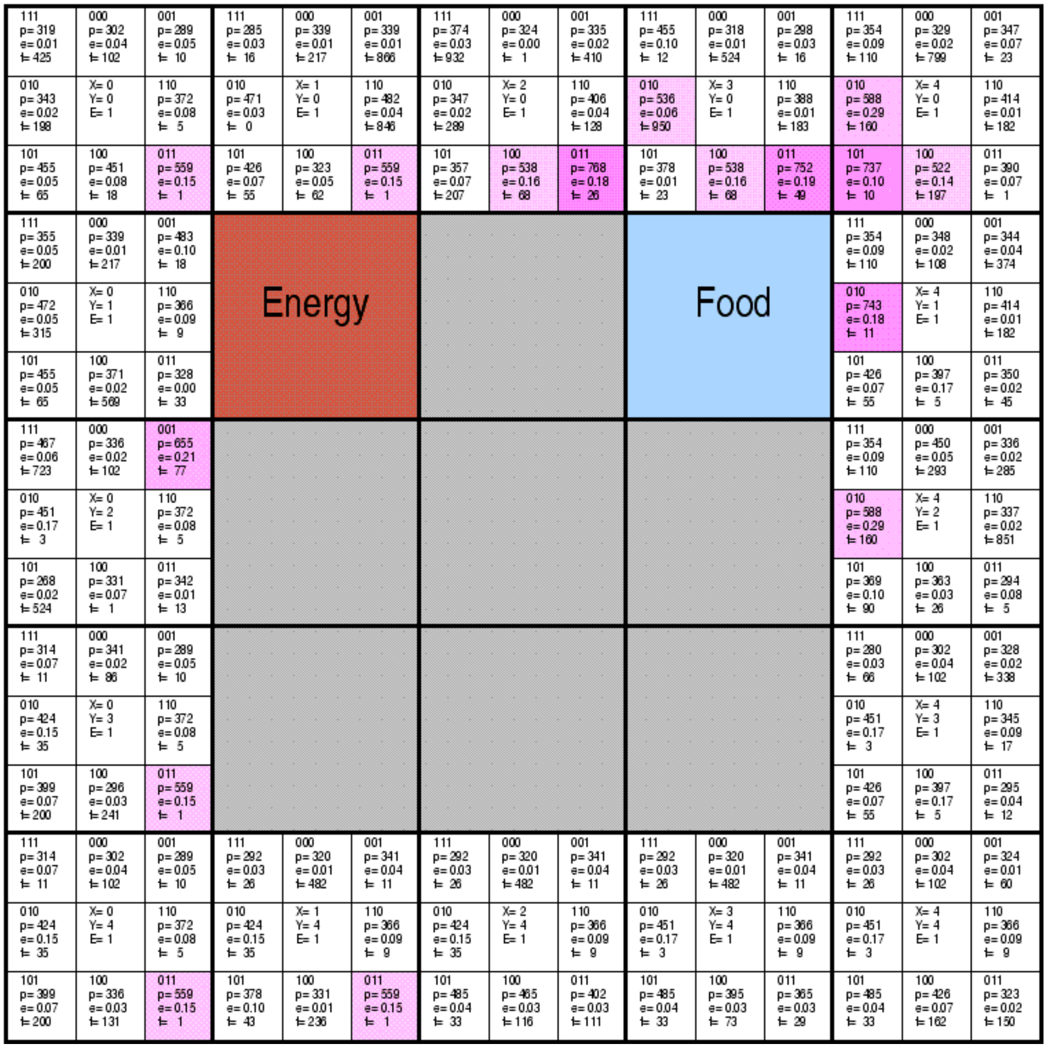
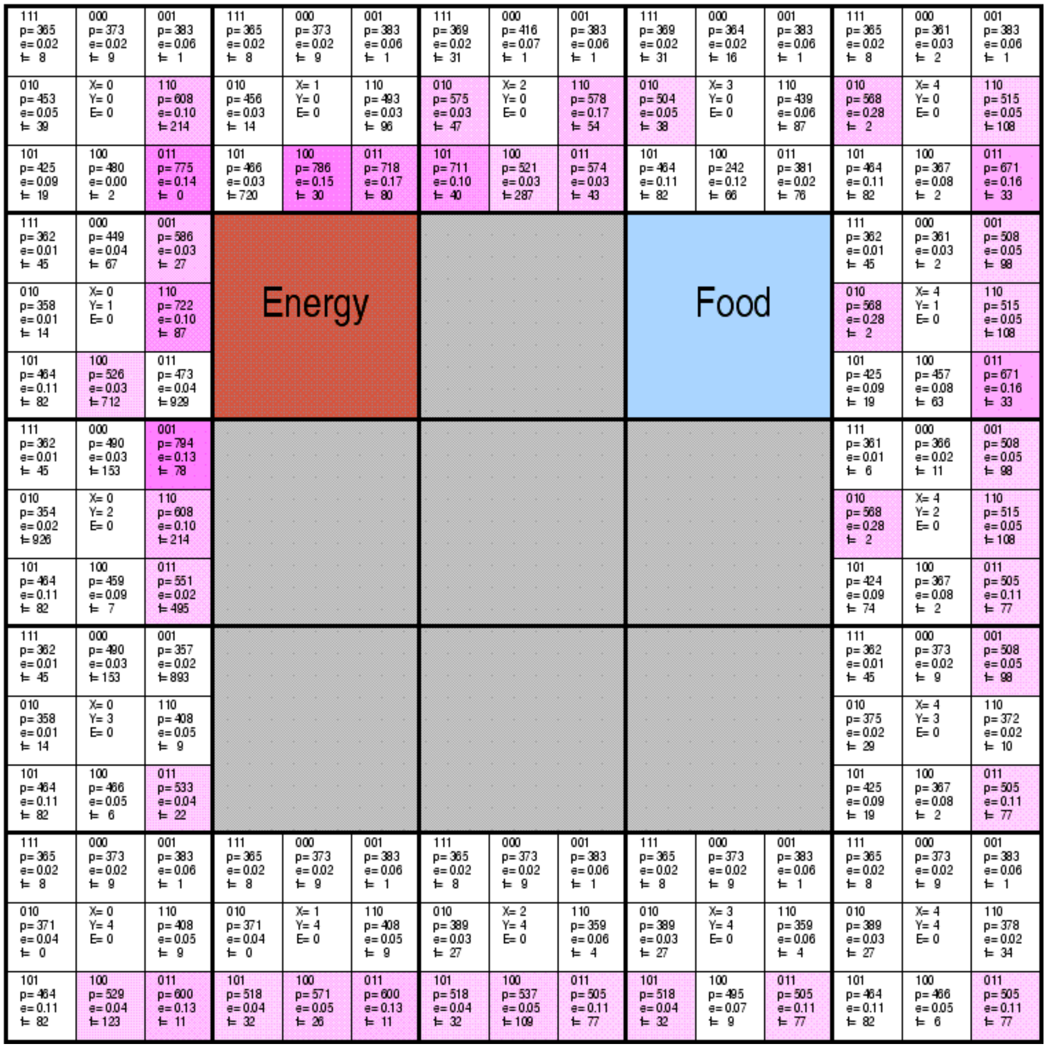


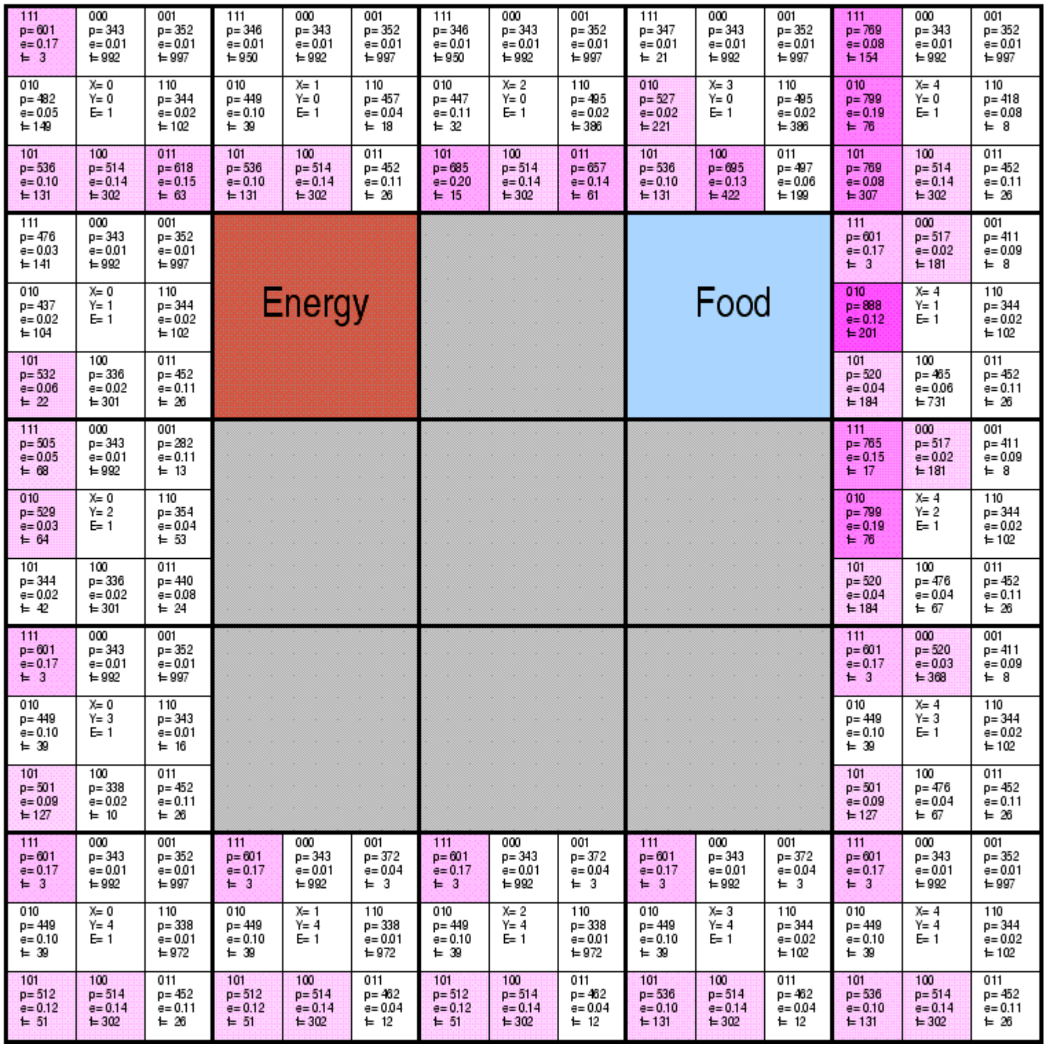
Figure 4-44 Random exploration. 'Food Trial'

Figure 4-45 and Figure 4-46 show the same diagrams generated using roulette selection in the exploration phase. There appears to be a much more even distribution of shaded boxes, indicating that the two halves of the problem; what to do when energy is low, and what to do when it is high, have both been solved equally well.



111 p=365 e=0.02 t= 8	000 p=373 e=0.02 t= 9	001 p=383 e=0.06 t= 1	111 p=365 e=0.02 t= 8	000 p=373 e=0.02 t= 9	001 p=383 e=0.06 t= 1	111 p=369 e=0.02 t= 31	000 p=416 e=0.07 t= 1	001 p=383 e=0.06 t= 1	111 p=369 e=0.02 t= 31	000 p=364 e=0.02 t= 16	001 p=383 e=0.06 t= 1	111 p=365 e=0.02 t= 8	000 p=361 e=0.03 t= 2	001 p=383 e=0.06 t= 1
010 p=453 e=0.05 t= 39	X= 0 Y= 0 E= 0	110 p=608 e=0.10 t=214	010 p=456 e=0.03 t= 14	X= 1 Y= 0 E= 0	110 p=493 e=0.03 t= 96	010 p=575 e=0.03 t= 47	X= 2 Y= 0 E= 0	110 p=578 e=0.17 t= 54	010 p=504 e=0.05 t= 38	X= 3 Y= 0 E= 0	110 p=439 e=0.06 t= 87	010 p=568 e=0.28 t= 2	X= 4 Y= 0 E= 0	110 p=515 e=0.05 t=108
101 p=425 e=0.09 t= 19	100 p=480 e=0.00 t= 2	011 p=775 e=0.14 t= 0	101 p=466 e=0.03 t=720	100 p=786 e=0.15 t= 30	011 p=718 e=0.17 t= 80	101 p=711 e=0.10 t= 40	100 p=521 e=0.03 t=287	011 p=574 e=0.03 t= 43	101 p=464 e=0.11 t= 82	100 p=242 e=0.12 t= 66	011 p=381 e=0.02 t= 76	101 p=464 e=0.11 t= 82	100 p=367 e=0.08 t= 2	011 p=671 e=0.16 t= 33
111 p=362 e=0.01 t= 45	000 p=449 e=0.04 t= 67	001 p=586 e=0.03 t= 27										111 p=362 e=0.01 t= 45	000 p=361 e=0.03 t= 2	001 p=508 e=0.05 t= 98
010 p=358 e=0.01 t= 14	X= 0 Y= 1 E= 0	110 p=722 e=0.10 t= 87		Energy						Food		010 p=568 e=0.28 t= 2	X= 4 Y= 1 E= 0	110 p=515 e=0.05 t=108
101 p=464 e=0.11 t= 82	100 p=526 e=0.03 t=712	011 p=473 e=0.04 t=929										101 p=425 e=0.09 t= 19	100 p=457 e=0.08 t= 63	011 p=671 e=0.16 t= 33
111 p=362 e=0.01 t= 45	000 p=490 e=0.03 t=153	001 p=794 e=0.13 t= 78										111 p=361 e=0.01 t= 6	000 p=366 e=0.02 t= 11	001 p=508 e=0.05 t= 98
010 p=354 e=0.02 t=926	X= 0 Y= 2 E= 0	110 p=608 e=0.10 t=214										010 p=568 e=0.28 t= 2	X= 4 Y= 2 E= 0	110 p=515 e=0.05 t=108
101 p=464 e=0.11 t= 82	100 p=459 e=0.09 t= 7	011 p=551 e=0.02 t=495										101 p=424 e=0.09 t= 74	100 p=367 e=0.08 t= 2	011 p=505 e=0.11 t= 77
111 p=362 e=0.01 t= 45	000 p=490 e=0.03 t=153	001 p=357 e=0.02 t=893										111 p=362 e=0.01 t= 45	000 p=373 e=0.02 t= 9	001 p=508 e=0.05 t= 98
010 p=358 e=0.01 t= 14	X= 0 Y= 3 E= 0	110 p=408 e=0.05 t= 9										010 p=375 e=0.02 t= 29	X= 4 Y= 3 E= 0	110 p=372 e=0.02 t= 10
101 p=464 e=0.11 t= 82	100 p=466 e=0.05 t= 6	011 p=533 e=0.04 t= 22										101 p=425 e=0.09 t= 19	100 p=367 e=0.08 t= 2	011 p=505 e=0.11 t= 77
111 p=365 e=0.02 t= 8	000 p=373 e=0.02 t= 9	001 p=383 e=0.06 t= 1	111 p=365 e=0.02 t= 8	000 p=373 e=0.02 t= 9	001 p=383 e=0.06 t= 1	111 p=365 e=0.02 t= 8	000 p=373 e=0.02 t= 9	001 p=383 e=0.06 t= 1	111 p=365 e=0.02 t= 8	000 p=373 e=0.02 t= 9	001 p=383 e=0.06 t= 1	111 p=365 e=0.02 t= 8	000 p=373 e=0.02 t= 9	001 p=383 e=0.06 t= 1
010 p=371 e=0.04 t= 0	X= 0 Y= 4 E= 0	110 p=408 e=0.05 t= 9	010 p=371 e=0.04 t= 0	X= 1 Y= 4 E= 0	110 p=408 e=0.05 t= 9	010 p=389 e=0.03 t= 27	X= 2 Y= 4 E= 0	110 p=359 e=0.06 t= 4	010 p=389 e=0.03 t= 27	X= 3 Y= 4 E= 0	110 p=359 e=0.06 t= 4	010 p=389 e=0.03 t= 27	X= 4 Y= 4 E= 0	110 p=378 e=0.02 t= 34
101 p=464 e=0.11 t= 82	100 p=529 e=0.04 t=123	011 p=600 e=0.13 t= 11	101 p=518 e=0.04 t= 32	100 p=571 e=0.05 t= 26	011 p=600 e=0.13 t= 11	101 p=518 e=0.04 t= 32	100 p=537 e=0.05 t=109	011 p=505 e=0.11 t= 77	101 p=518 e=0.04 t= 32	100 p=495 e=0.07 t= 9	011 p=505 e=0.11 t= 77	101 p=464 e=0.11 t= 82	100 p=466 e=0.05 t= 6	011 p=505 e=0.11 t= 77

Figure 4-45 Roulette exploration. 'Energy Trial'



111 p=601 e=0.17 t= 3	000 p=343 e=0.01 t=992	001 p=352 e=0.01 t=997	111 p=346 e=0.01 t=950	000 p=343 e=0.01 t=992	001 p=352 e=0.01 t=997	111 p=346 e=0.01 t=950	000 p=343 e=0.01 t=992	001 p=352 e=0.01 t=997	111 p=347 e=0.01 t= 21	000 p=343 e=0.01 t=992	001 p=352 e=0.01 t=997	111 p=769 e=0.08 t=154	000 p=343 e=0.01 t=992	001 p=352 e=0.01 t=997
010 p=482 e=0.05 t=149	X=0 Y=0 E=1	110 p=344 e=0.02 t=102	010 p=449 e=0.10 t= 39	X=1 Y=0 E=1	110 p=457 e=0.04 t= 18	010 p=447 e=0.11 t= 32	X=2 Y=0 E=1	110 p=495 e=0.02 t=386	010 p=527 e=0.02 t=221	X=3 Y=0 E=1	110 p=495 e=0.02 t=386	010 p=799 e=0.19 t= 76	X=4 Y=0 E=1	110 p=418 e=0.08 t= 8
101 p=536 e=0.10 t=131	100 p=514 e=0.14 t=302	011 p=618 e=0.15 t= 63	101 p=536 e=0.10 t=131	100 p=514 e=0.14 t=302	011 p=452 e=0.11 t= 26	101 p=685 e=0.20 t= 15	100 p=514 e=0.14 t=302	011 p=657 e=0.14 t= 61	101 p=536 e=0.10 t=131	100 p=695 e=0.13 t=422	011 p=497 e=0.06 t=199	101 p=769 e=0.08 t=307	100 p=514 e=0.14 t=302	011 p=452 e=0.11 t= 26
111 p=476 e=0.03 t=141	000 p=343 e=0.01 t=992	001 p=352 e=0.01 t=997										111 p=601 e=0.17 t= 3	000 p=517 e=0.02 t=181	001 p=411 e=0.09 t= 8
010 p=437 e=0.02 t=104	X=0 Y=1 E=1	110 p=344 e=0.02 t=102		Energy						Food		010 p=888 e=0.12 t=201	X=4 Y=1 E=1	110 p=344 e=0.02 t=102
101 p=532 e=0.06 t= 22	100 p=336 e=0.02 t=301	011 p=452 e=0.11 t= 26										101 p=520 e=0.04 t=184	100 p=465 e=0.06 t=731	011 p=452 e=0.11 t= 26
111 p=505 e=0.05 t= 68	000 p=343 e=0.01 t=992	001 p=282 e=0.11 t= 13										111 p=765 e=0.15 t= 17	000 p=517 e=0.02 t=181	001 p=411 e=0.09 t= 8
010 p=529 e=0.03 t= 64	X=0 Y=2 E=1	110 p=354 e=0.04 t= 53										010 p=799 e=0.19 t= 76	X=4 Y=2 E=1	110 p=344 e=0.02 t=102
101 p=344 e=0.02 t= 42	100 p=336 e=0.02 t=301	011 p=440 e=0.08 t= 24										101 p=520 e=0.04 t=184	100 p=476 e=0.04 t= 67	011 p=452 e=0.11 t= 26
111 p=601 e=0.17 t= 3	000 p=343 e=0.01 t=992	001 p=352 e=0.01 t=997										111 p=601 e=0.17 t= 3	000 p=520 e=0.03 t=368	001 p=411 e=0.09 t= 8
010 p=449 e=0.10 t= 39	X=0 Y=3 E=1	110 p=343 e=0.01 t= 16										010 p=449 e=0.10 t= 39	X=4 Y=3 E=1	110 p=344 e=0.02 t=102
101 p=501 e=0.09 t=127	100 p=338 e=0.02 t= 10	011 p=452 e=0.11 t= 26										101 p=501 e=0.09 t=127	100 p=476 e=0.04 t= 67	011 p=452 e=0.11 t= 26
111 p=601 e=0.17 t= 3	000 p=343 e=0.01 t=992	001 p=352 e=0.01 t=997	111 p=601 e=0.17 t= 3	000 p=343 e=0.01 t=992	001 p=372 e=0.04 t= 3	111 p=601 e=0.17 t= 3	000 p=343 e=0.01 t=992	001 p=372 e=0.04 t= 3	111 p=601 e=0.17 t= 3	000 p=343 e=0.01 t=992	001 p=372 e=0.04 t= 3	111 p=601 e=0.17 t= 3	000 p=343 e=0.01 t=992	001 p=352 e=0.01 t=997
010 p=449 e=0.10 t= 39	X=0 Y=4 E=1	110 p=338 e=0.01 t=972	010 p=449 e=0.10 t= 39	X=1 Y=4 E=1	110 p=338 e=0.01 t=972	010 p=449 e=0.10 t= 39	X=2 Y=4 E=1	110 p=338 e=0.01 t=972	010 p=449 e=0.10 t= 39	X=3 Y=4 E=1	110 p=344 e=0.02 t=102	010 p=449 e=0.10 t= 39	X=4 Y=4 E=1	110 p=344 e=0.02 t=102
101 p=512 e=0.12 t= 51	100 p=514 e=0.14 t=302	011 p=452 e=0.11 t= 26	101 p=512 e=0.12 t= 51	100 p=514 e=0.14 t=302	011 p=462 e=0.04 t= 12	101 p=512 e=0.12 t= 51	100 p=514 e=0.14 t=302	011 p=462 e=0.04 t= 12	101 p=536 e=0.10 t=131	100 p=514 e=0.14 t=302	011 p=462 e=0.04 t= 12	101 p=536 e=0.10 t=131	100 p=514 e=0.14 t=302	011 p=452 e=0.11 t= 26

Figure 4-46 Roulette exploration. 'Food Trial'

It appears then that XCS may be able to concentrate resources better using roulette action selection in the exploration phase when these resources are a limiting factor in its performance; i.e. *N* is too small.

This problem (Woods1e-type3) in which the seemingly anomalous results occur is a difficult one for XCS to solve. The reward varies in direct proportion to the animat's internal 'energy level' when it is at a goal state. There will thus be a high degree of error in all predictions. Perhaps the results observed in this continuous reward problem reveal an effect of low values of *N* that have been masked in the other, less challenging problems?

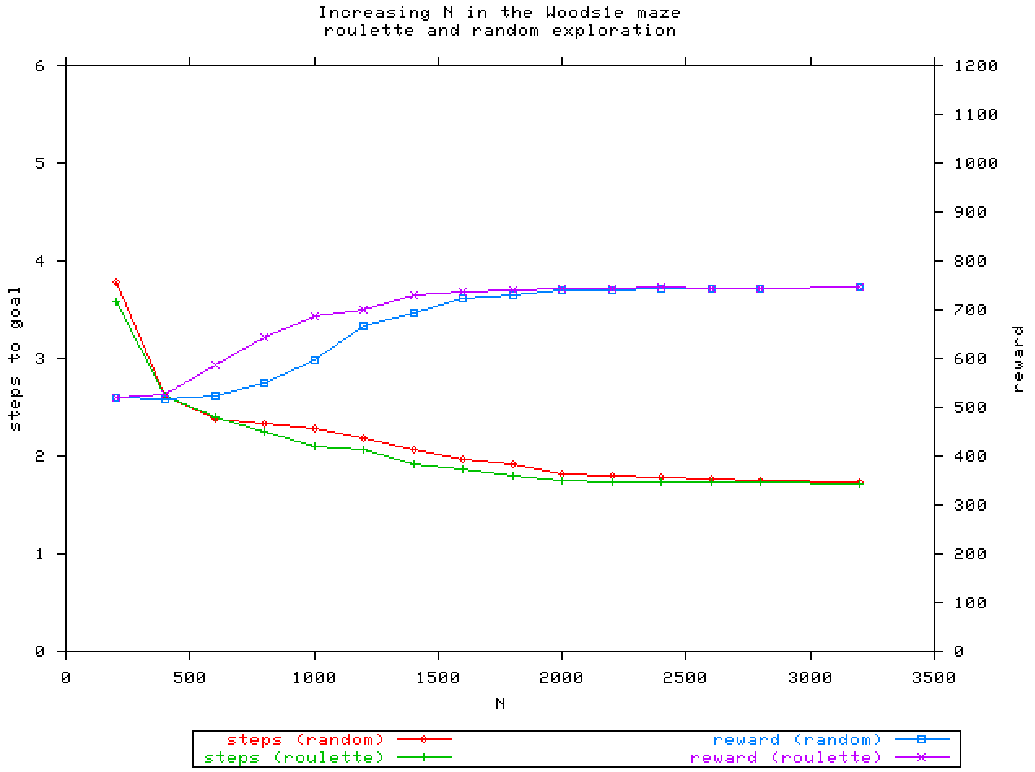


Figure 4-47 Woods1e-type3. Effect of increasing *N* with roulette and random exploration (continuous reward, cost of movement).

In Figure 4-47 we see the effect of *N* on the performance of XCS in the Woods1e-type3 problem, using the two exploration action-selection strategies. It will be seen that at low values of *N*, roulette selection strongly out-performs random exploration.

N	*t Stat*	*t Crit*
200	-0.107	2.110
400	0.090	2.145
600	12.650	2.101
800	16.579	2.101
1000	16.216	2.131
1200	9.420	2.101
1400	7.050	2.101
1600	0.854	2.101
1800	0.890	2.110
2000	0.780	2.131
2200	0.444	2.101
2400	0.260	2.101
2600	0.470	2.101
2800	0.160	2.101
3000	0.370	2.101

Table 4-9 Two-tailed Student t-Test for hypothesised zero difference between reward gained using roulette and random action selection, for increasing values of N. Significantly different region shaded.

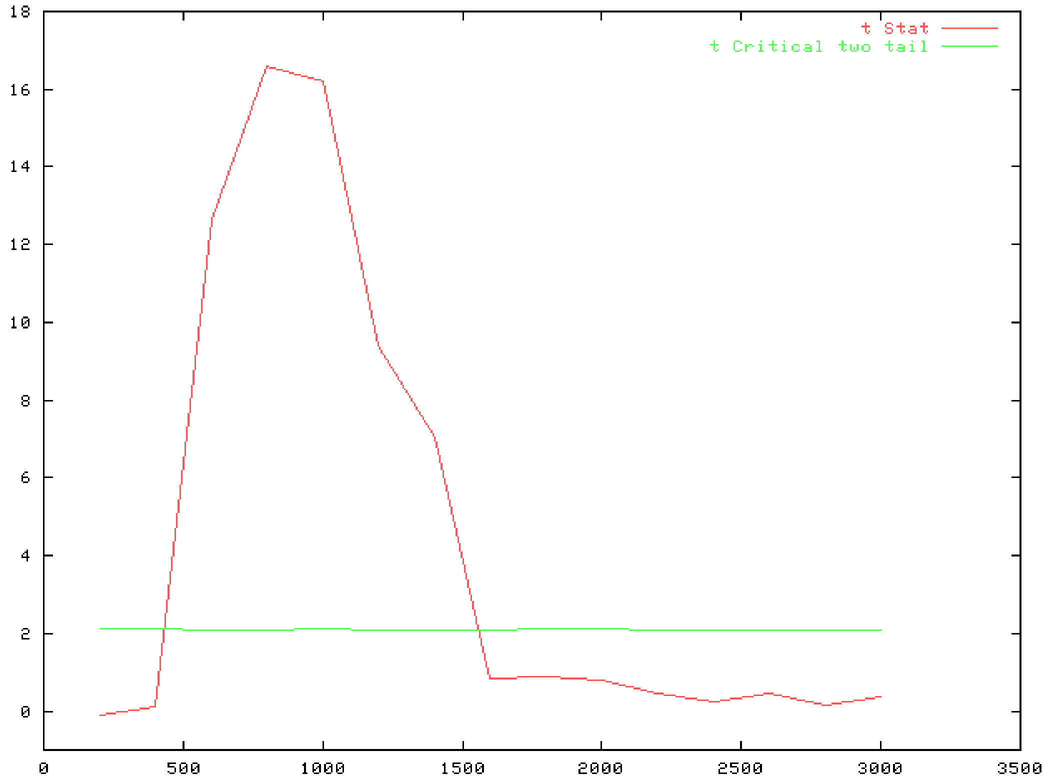


Figure 4-48 Graph of Table 4-8, showing area of significant difference (above the t Critical line).

The same is true in the simpler Woods1e-type1 problem, as shown in Figure 4-49. Note that the advantages of roulette action selection are reduced as *N* rises, until a random exploration policy is better. This eventual benefit of random exploration is explained by the tendency of XCS with roulette selection to over-generalise, as shown earlier with Maze5em.

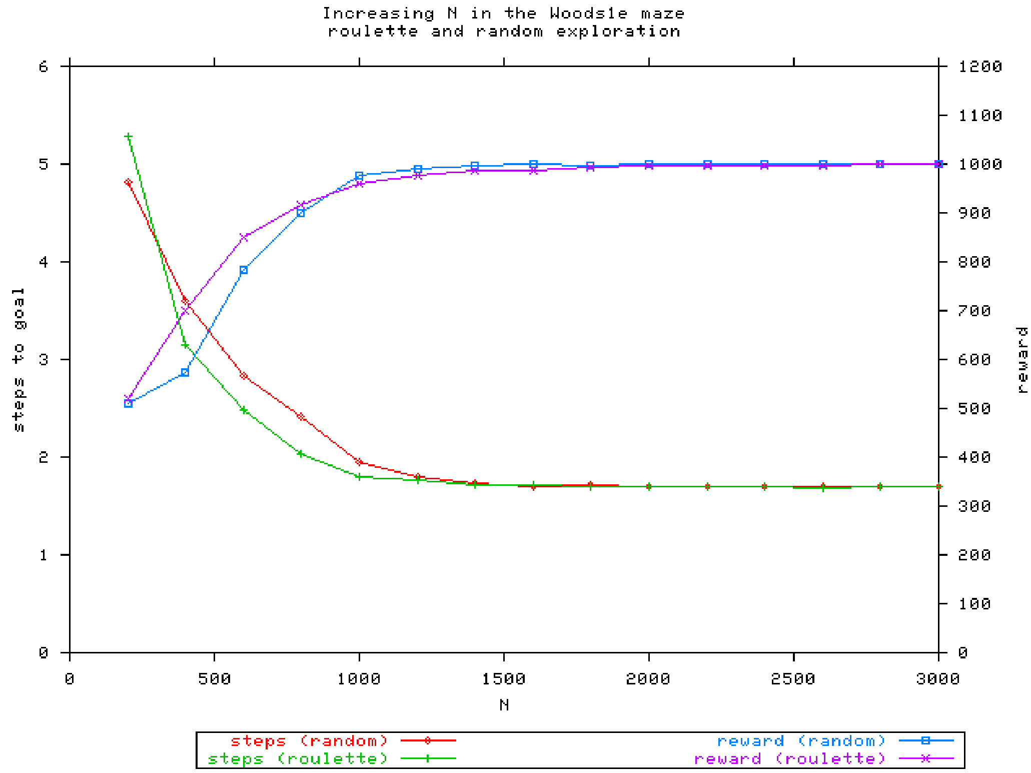


Figure 4-49 Woods1e-type1. Effect of increasing *N* with roulette and random exploration (stepwise reward, free movement).

How then can we explain the advantages shown by roulette selection at low values of *N*? It was suggested above that using roulette selection, XCS was able to concentrate scarce resources on the high value areas of the problem space, thus maximising its overall returns. If resources are scarce, roulette selection can only give this performance advantage if classifiers that predict a lower payoff can be deleted.

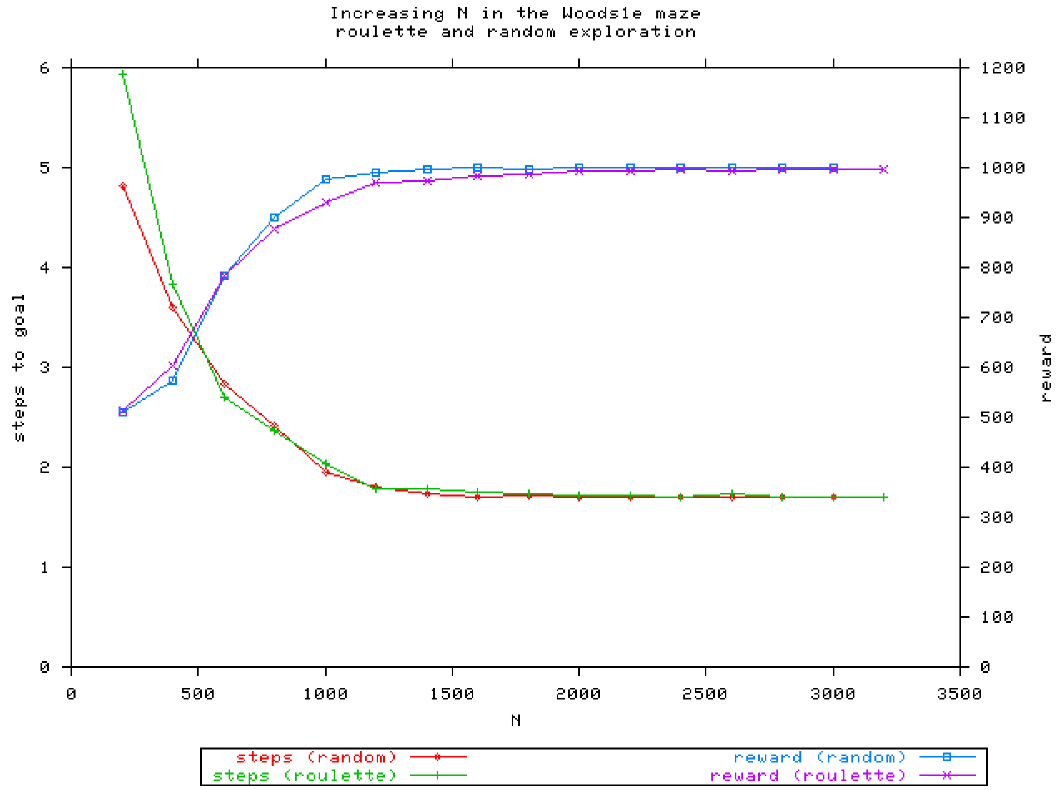


Figure 4-50 Without fitness reduction in offspring, roulette's advantage at lower *N* is reduced.

In XCS there is typically a reduction in fitness levied on the offspring of classifiers newly produced by the GA. Figure 4-50 shows that when this fitness reduction does not occur, roulette wheel exploration ceases to give such a marked advantage at lower values of *N* for the Woods1e-type1 problem. This supports the hypothesis that roulette wheel exploration provides a performance advantage by allowing the system to grab resources from lower payoff niches. In less visited niches, offspring would normally have their fitness reduced (as indeed would offspring in all niches). Because the niche is less often visited, they have little chance to boost their fitness by participation in [A] (in any case, fitness increases slowly since MAM does not operate on fitness updates.). These lower fitness classifiers then become candidates for deletion when the GA is triggered in a more frequently visited state. In this way resources are 'stolen' from low payoff state-action niches, since the roulette-wheel exploration strategy increases the bias towards visiting the higher payoff niches.

We can relate these findings to the pressures and challenges recognised by Butz at al in [Butz2003, Butz2003b]. Figure 4-51 shows two areas where different challenges face XCS. In Area 1, it is suggested that a new challenge is operating, which will be termed the '*niche resource challenge*'. At values of *N* insufficient to satisfy the ROP-bound, the roulette wheel exploration strategy benefits performance through 'resource theft', as indicated above.

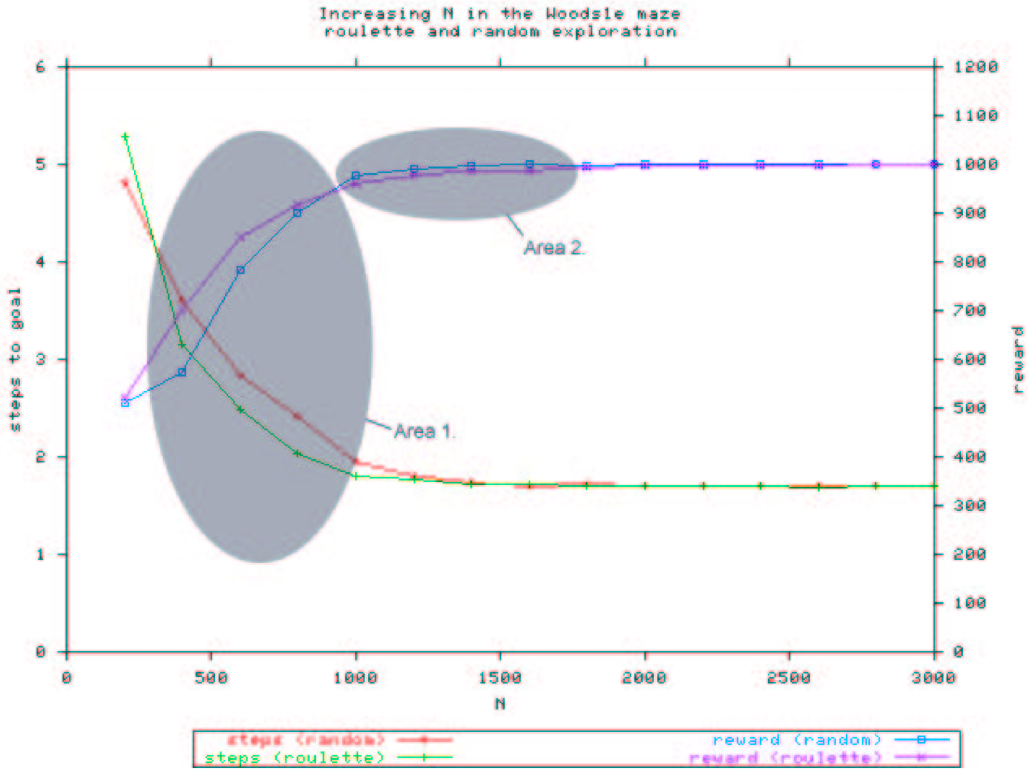


Figure 4-51 At different values of *N*, XCS faces different challenges

Figure 4-52 indicates that the sub-optimal performance of XCS is not due to the '*cover challenge*', since for all values of *N*, cover occurs infrequently and has stopped after trial 2000 of the total 15000 exploit trials in each experiment.

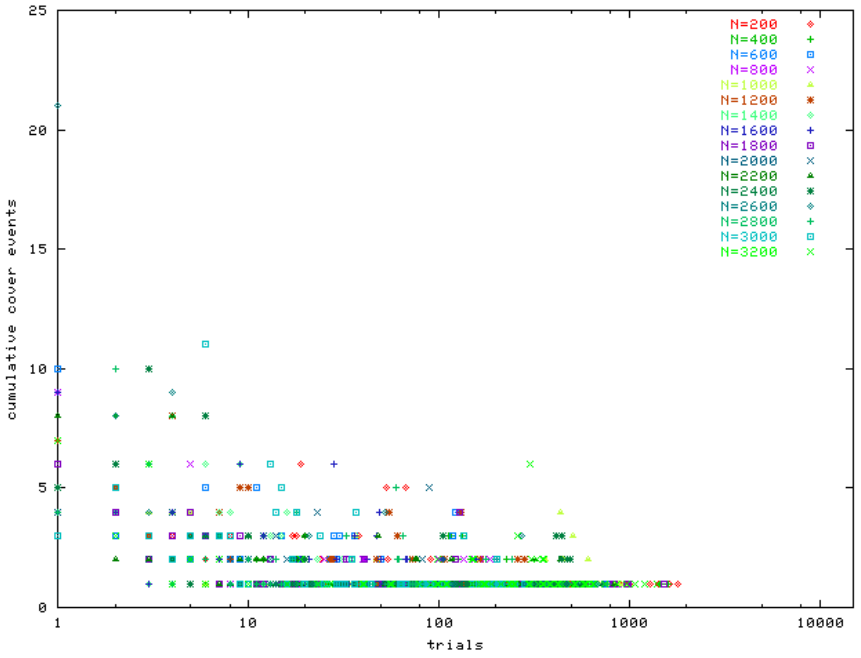


Figure 4-52 Frequency of covering against trial for different values of *N* - random action selection, Woods1e-type1.

In Area 2, it is suggested that the sub-optimal behaviour using roulette-wheel exploration is evidence of the effect of the bias on the '*reproductive opportunity bound*' of Butz et al. [Butz2003a]. As previously stated, the ROP-bound is derived on the assumption that all possible states in the space $\{0, 1\}^l$ will be uniformly presented to the system. All other things being equal, this will be true if the exploration policy does not introduce any bias[15].

There is a bias in exploration when using roulette-wheel action selection; fitter classifiers will be picked more often than less fit ones, and this bias will increase as more trials are performed. Therefore new candidate solutions in lower-payoff niches will be more likely to be deleted when using roulette exploration. It seems that this might explain the necessity to increase *N* in order to get equivalent optimality to that achieved with random exploration – the dilution effect reduces the probability of deletion. The ROP-bound is dependent upon the exploration policy.

[15] This theoretical work by Butz et al considers only single step environments. Clearly, in multi-step environments a further bias is created as states nearer the goal will be sampled more frequently than states further away.

4.7 Conclusions

In this chapter we have seen that XCS can optimally solve a variety of simple multi-objective tasks. It can do these using a variety of action-selection policies in the exploration phase. In the results presented here, we have seen that;

- Although XCS has many parameters, these seem to need little adjustment to function (nearly) optimally. This is unlike e.g. ZCS [Wilson1994] in which optimal performance has been shown [Bull2002a] to be strongly dependent on the values of the reinforcement learning parameters. In XCS, optimality is the norm rather than the exception with respect to the settings of β and γ.
- XCS can be used successfully with roulette-wheel exploration replacing random action selection, suggesting that it may be used for on-line learning where random actions would be inappropriate.
- When population sizes are too small for random action selection to perform optimally, roulette-wheel action selection may give a performance benefit. This appears to be due to the advantages of concentrating resources on high payoff niches where there are insufficient resources to form a complete and accurate map of state-action pairs. This may be of value in a robotic environment; smaller populations are less computationally expensive. Within this thesis, this has been termed the '*niche resource challenge*'.
- Roulette-wheel action selection functions less well than a random action-selection policy at population sizes sufficient for optimal performance using random action selection. This appears to be due to over-generality resulting from a lack of exploration, and adds weight to the observations of Lanzi [Lanzi1999] and the theoretical work of Butz et al. [Butz2003a]

- There is evidence for the ROP-bound of Butz et al. in multi-step environments, but it is dependent on the action-selection policy.

Chapter 5 X-TCS

5.1 Introduction

We have seen that ZCS, a strength-based LCS, is capable of optimal performance in simulated problems with multiple goals. However, optimal performance can only be achieved after much careful adjustment of the reinforcement learning parameters. TCS, an adapted version of ZCS which automatically determines the change in environmental state that can be considered significant and which applies reinforcement according to the time taken both to achieve reward and for individual actions was used to control a real robot. Although TCS was able to approach optimal behaviour on a single objective task, it was not possible to demonstrate reliably optimal behaviour with two objectives.

In contrast to the parameter sensitivity of ZCS, the accuracy-based XCS requires little adjustment of its reinforcement learning parameters in order to achieve optimality in the same simulated tasks. It was suggested that XCS could be made suitable for online control of a robot by substituting a probabilistic action selection policy for the random action selection policy typically used in ‘explore’ trials, and noted that there is some evidence that using such a policy may enable better performance with smaller population sizes as might be necessary in a resource-limited environment such as a physical robot.

This chapter describes an extension of XCS that incorporates the changes that Hurst et al. made to ZCS in order to produce TCS [Hurst2002a, Hurst2002b, Hurst 2003]. The new algorithm, X-TCS, is set the same tasks of controlling a physical robot as described in Chapter 3. Experimental results are presented which show convincing performance improvements over TCS.

To the best of the author's knowledge, this is the first application of an accuracy-based LCS to controlling a physical agent in the real world **without *a priori* discretisation**.

5.2 Algorithmic Description

The changes made to XCS to produce X-TCS are a parallel of those described in Section 3.3.3. XCS forms a match set, and thence an action set using either a deterministic exploit policy or a roulette-wheel explore policy. The action is then initiated and the algorithm enters the drop decision cycle, in which it is determined whether to carry on with the current action or stop and create a new match set. The process continues until external reinforcement is attained or some time-out value for the trial is reached. This is pictorially represented in Figure 3-2. Rather than basing the drop decision on the fitness of the classifiers in drop and continue sets, X-TCS uses the values in the prediction array. In this way the drop decision is reward-based as is the fitness-based decision in TCS.

In addition to the use of the drop decision cycle to discover what scale of environmental change may optimally be regarded as 'significant' in different parts of the environment, the update procedures of reinforcement learning are also modified. As in Section 3.3.3, the external reward is discounted according to the amount of time taken to reach the goal, and the reinforcement that flows from the classifiers that advocate an action to their predecessors is similarly discounted. As before, these alterations should favour efficiency of the solution as a whole, and the use of a smaller number of long actions rather than a large number of small actions.

5.3 Experimental set-up

To enable a useful comparison with the performance of TCS, the physical environment and robot platform are unchanged. Attainment of external goal states is through the monitoring of power supplies as before. The alterations to XCS additional to those outlined in the previous section are as in Section 3.4.3, and are listed briefly below, viz;

- Unordered pairs of real numbers are used to match the environmental input of the three LDRs. The condition of the classifiers is composed of three such pairs, one for each LDR.
- Classifiers have an action from the set {0, 1, 2} corresponding to drive left, forward or right.
- The genetic operators of crossover and mutation are altered such that crossover occurs with uniform probability within the condition of the classifier, and to allow mutation of a real number within the bounds [0, 1]
- Cover is altered to account for real-valued environmental input.

Although GA subsumption is used to compact the rule base and promote generalization, action set subsumption is not used since this would remove the variability in the action set upon which the drop decision cycle relies.

Apart from the addition of the drop decision cycle, the changes to the reinforcement regime to account for the time taken to perform actions, and the implementation details outlined above, X-TCS is identical to the description of XCS outlined in Section 4.2.

5.4 Results

As in the work using TCS presented in Chapter 3, a single objective problem was first used to demonstrate the ability of the system to learn. Thereafter, a two objective problem was used. Since the XCS policy of alternating explore and exploit trials was used all metrics are presented in three forms, these being only exploit trials, only explore trials, and the average of both on a per-trial basis.

All results are the average of five experiments, unless otherwise stated. The total number of trials performed is comparable to that presented for TCS, i.e. where 200 TCS trials were performed, 100 explore and 100 exploit trials were performed with X-TCS. Where a windowed average is shown, the window size is therefore reduced to 20 trials rather than 50 as used for the TCS results.

5.4.1 One Objective

The single objective task is identical to that reported in Section 3.5.1. As before, external reinforcement upon reaching the goal state is 1000 before time-proportionate discounting is applied. Trial time-out occurs after 30 seconds.

Figure 5-1, Figure 5-2 and Figure 5-3 show the number of actions taken by X-TCS to reach the goal state and gain external reinforcement. It can be seen that in each case there is a trend for the number of actions taken to diminish with time, showing that the system is finding more efficient paths to the goal.

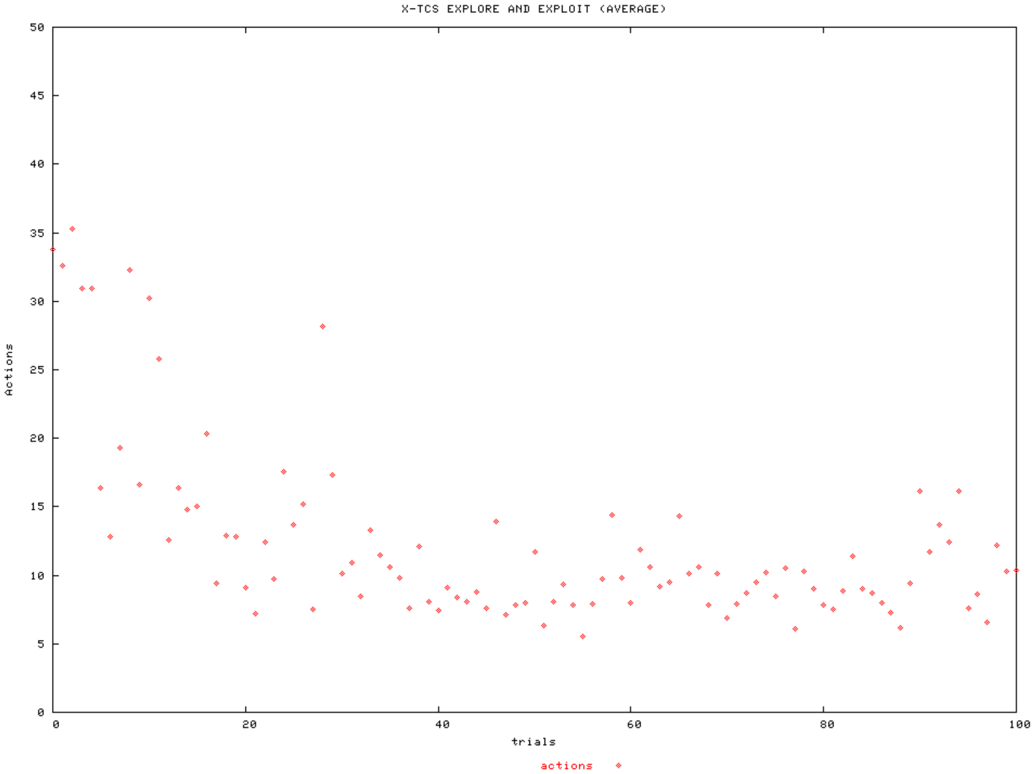


Figure 5-1 Actions Taken. Explore and Exploit

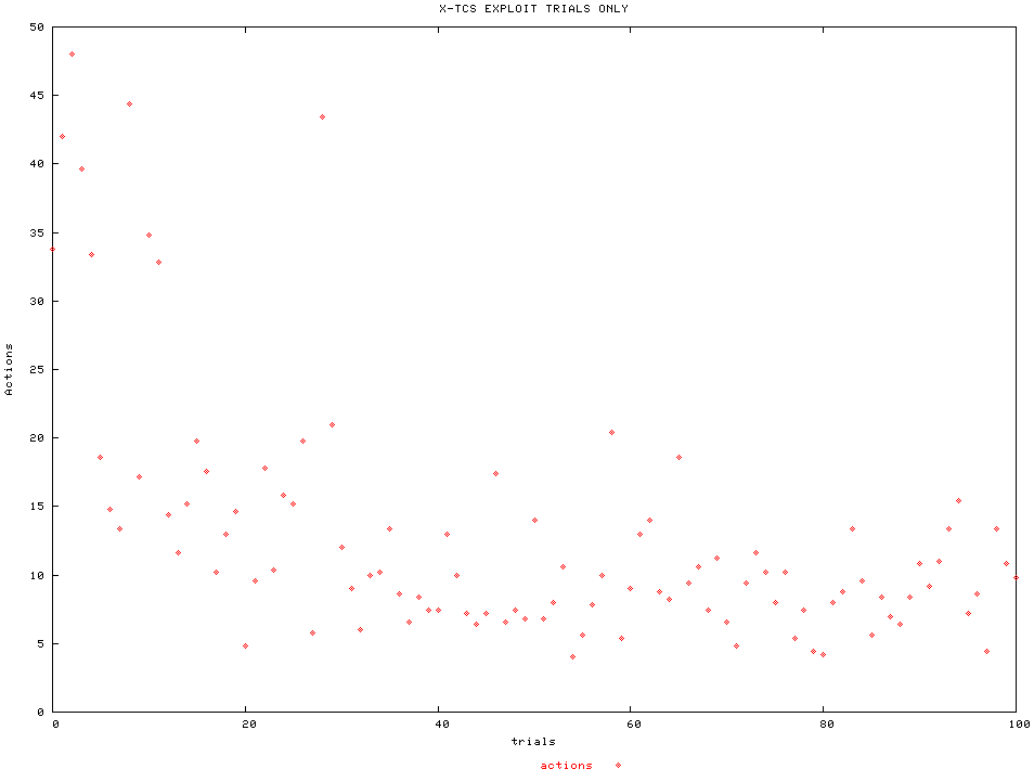


Figure 5-2 Actions Taken. Exploit only.

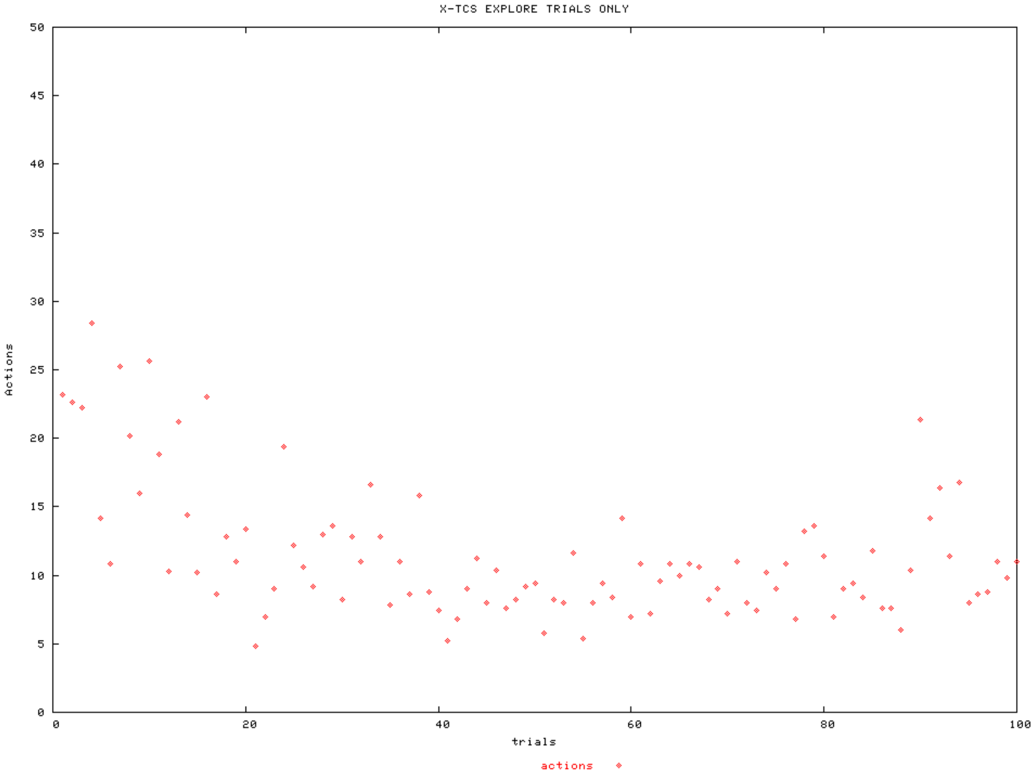


Figure 5-3 Actions Taken. Explore only

Figure 5-4, Figure 5-5 and Figure 5-6 show the ratio of failure to success. As with TCS, a trial is counted as a failure in the single objective task if the time-out value of 30 seconds is reached without the robot arriving at the goal. It can be seen that there is a strong tendency for the system to fail less often as learning continues to take place. As learning continues, the explore trials fail slightly more than the exploit trials, as would be expected.

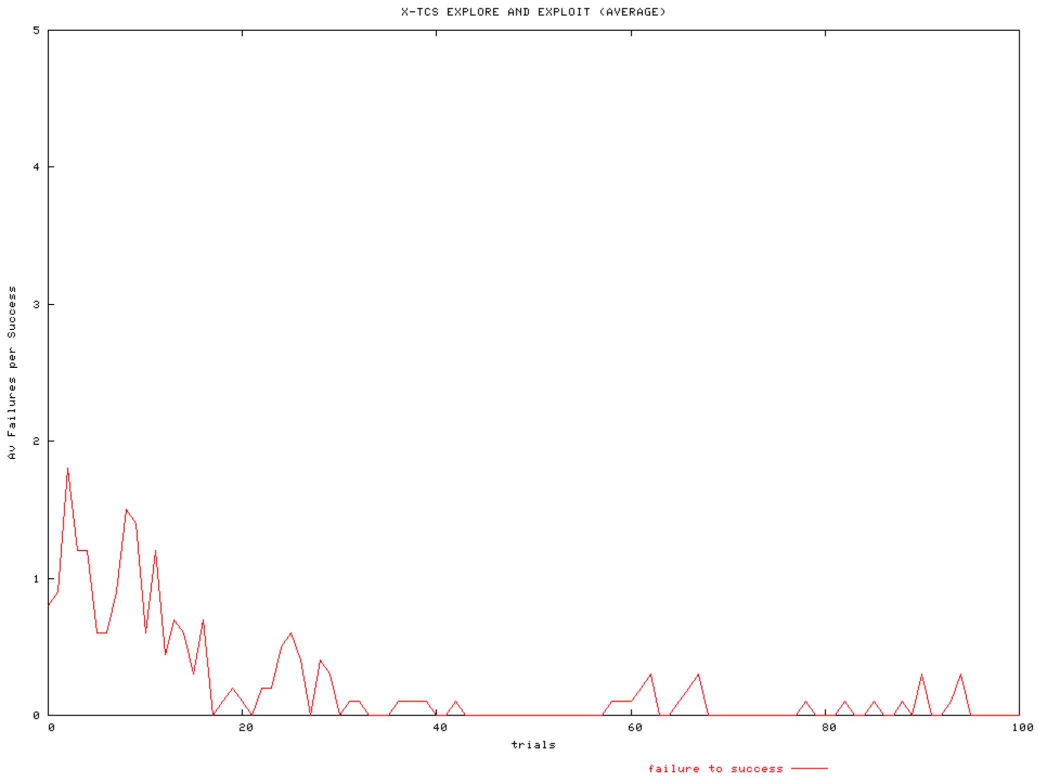


Figure 5-4 Failure to Success Ratio. Explore and Exploit

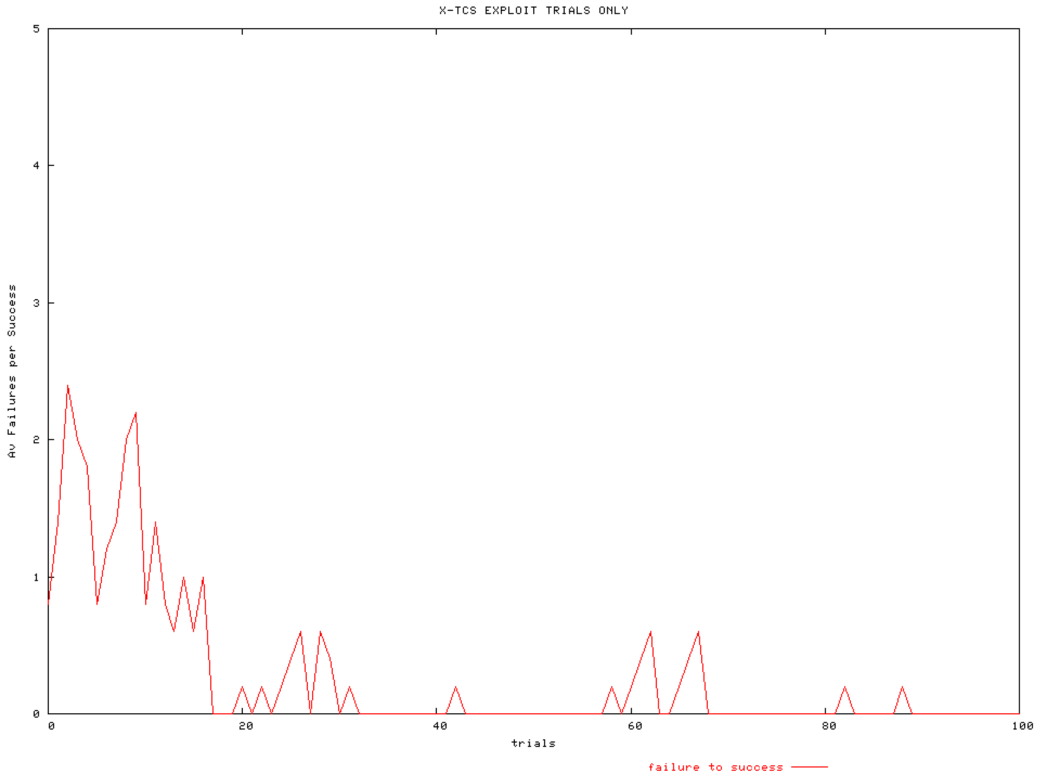


Figure 5-5 Failure to Success Ratio. Exploit only

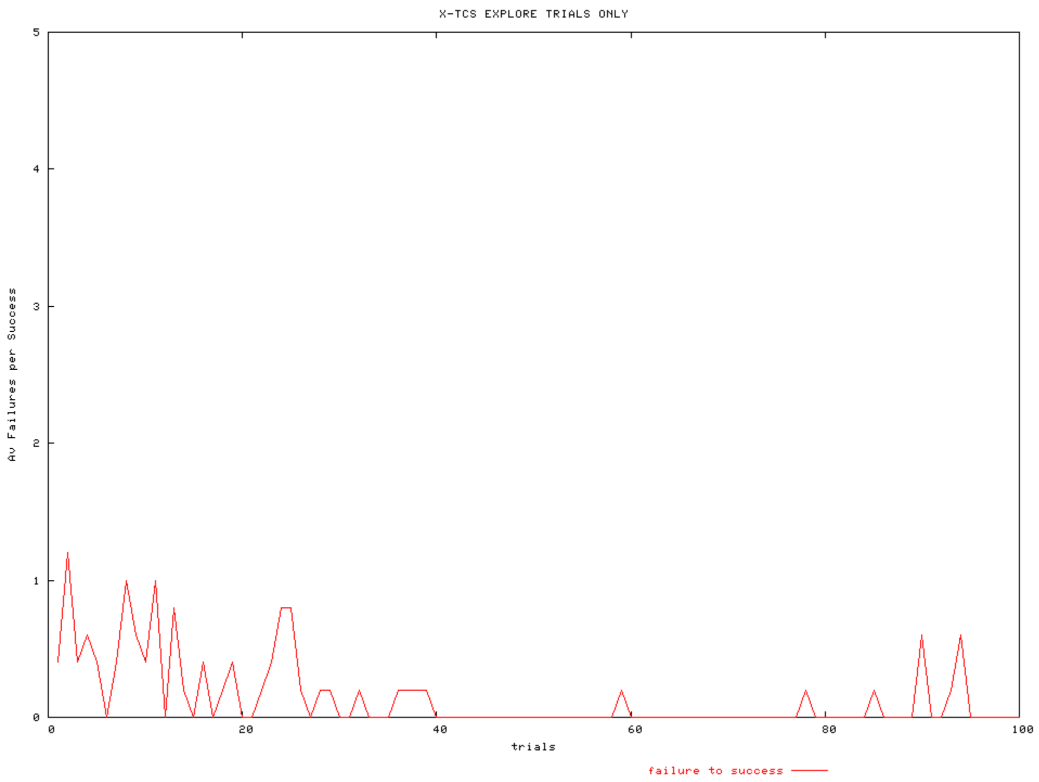


Figure 5-6 Failure to Success Ratio. Explore only

Figure 5-7, Figure 5-8, and Figure 5-9 show the total time taken to reach the goal in successful trials. It can be seen that there is a tendency for the time taken to decrease as learning takes place, falling towards the optimal case of between six and nine seconds. Again there is a visible difference between the performance in the explore and exploit trials – as expected, the exploit trials are both more nearly optimal with a mean of 10.8 compared with 11.7 for the explore trials, and are more consistently optimal with lower variance of 18.4 compared with 19.9.

Importantly, the average of the explore and exploit behaviour shows strong evidence of learning for all these metrics. Thus it would appear that X-TCS with a non-random explore policy is truly suited to online control.

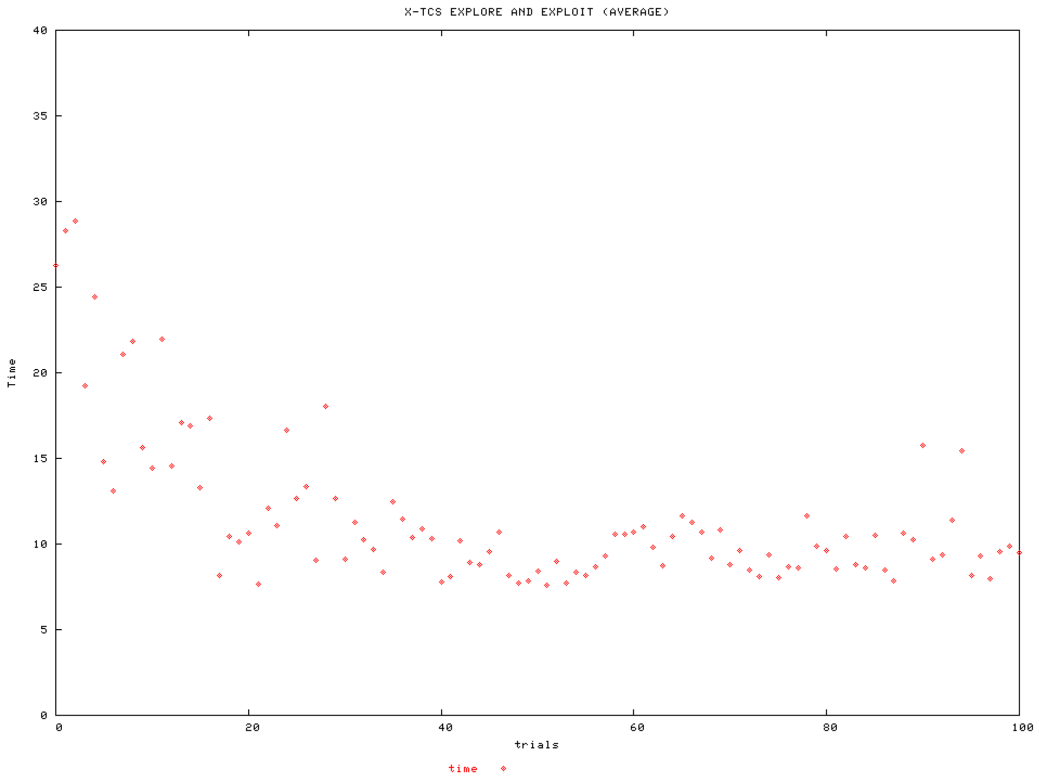


Figure 5-7 Time to Goal. Explore and Exploit

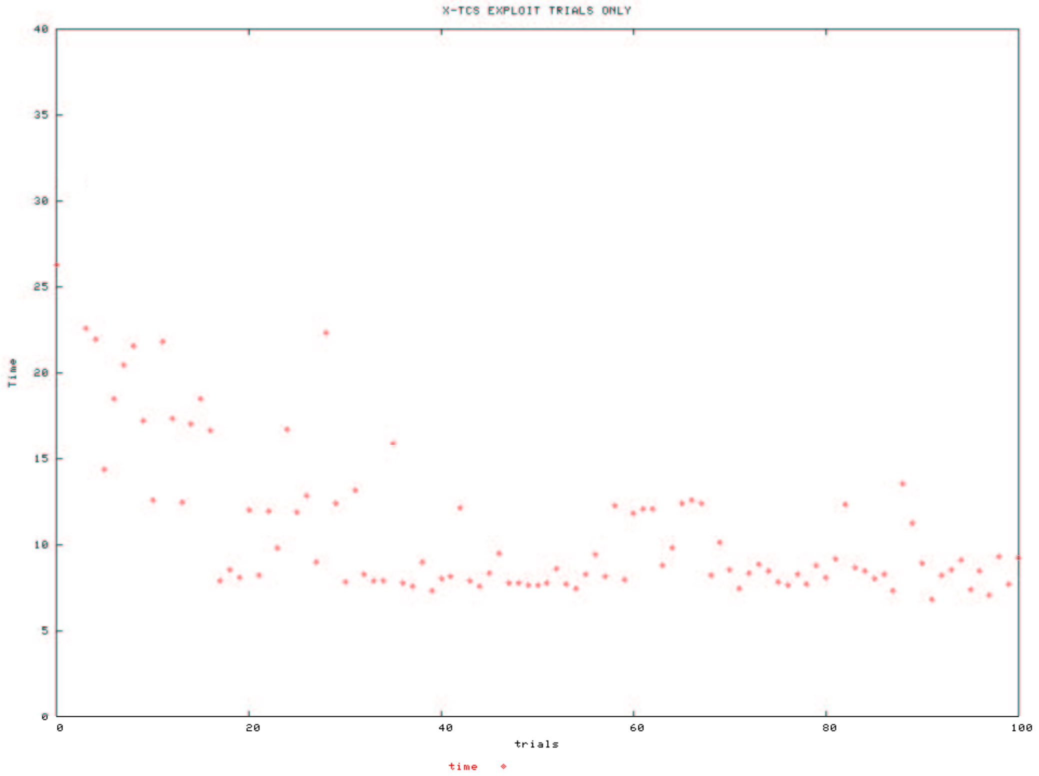


Figure 5-8 Time to Goal. Exploit only

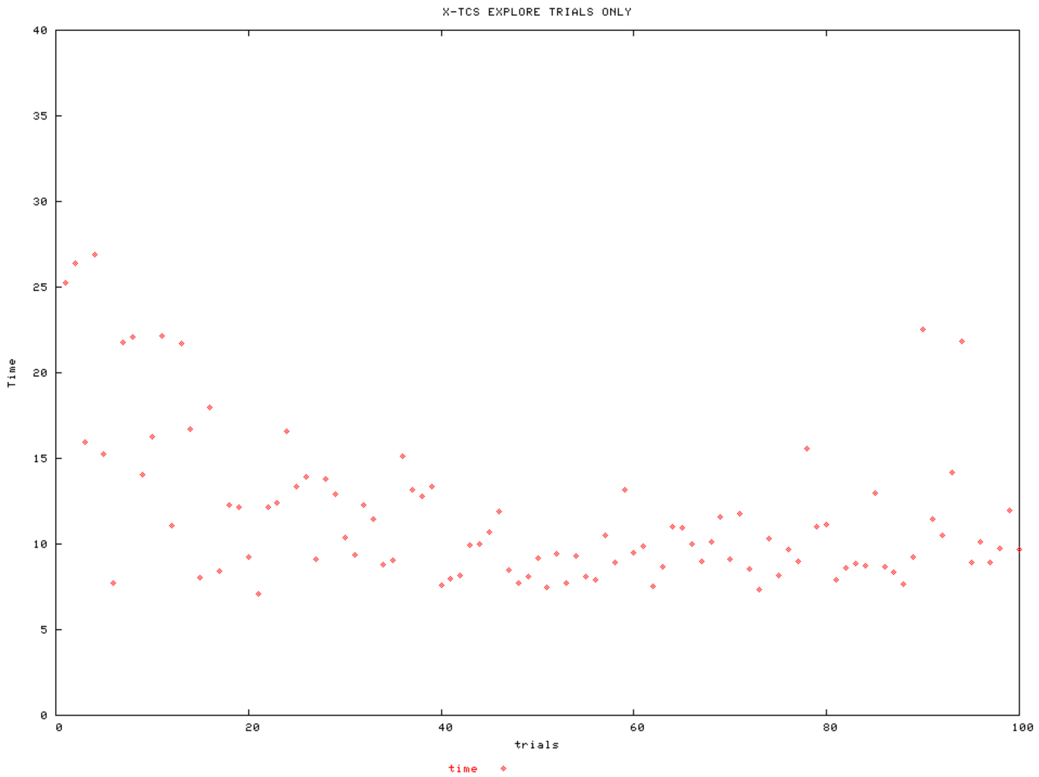


Figure 5-9 Time to Goal. Explore only

Figure 5-10, Figure 5-11 and Figure 5-12 show the windowed average of percentage success. As previously stated, the window size is 20 trials. Overall, the average performance of both exploit and explore trials shows a strong tendency towards the optimum, although neither exploit or explore trials on average were 100% successful.

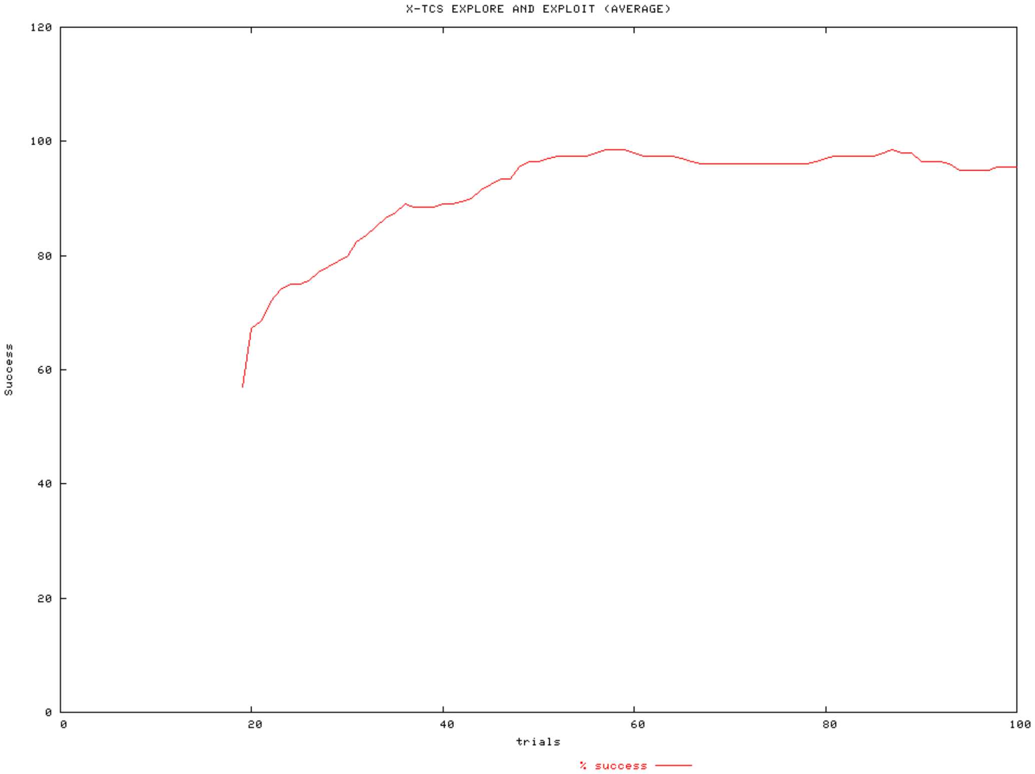


Figure 5-10 Windowed average % success. Explore and Exploit

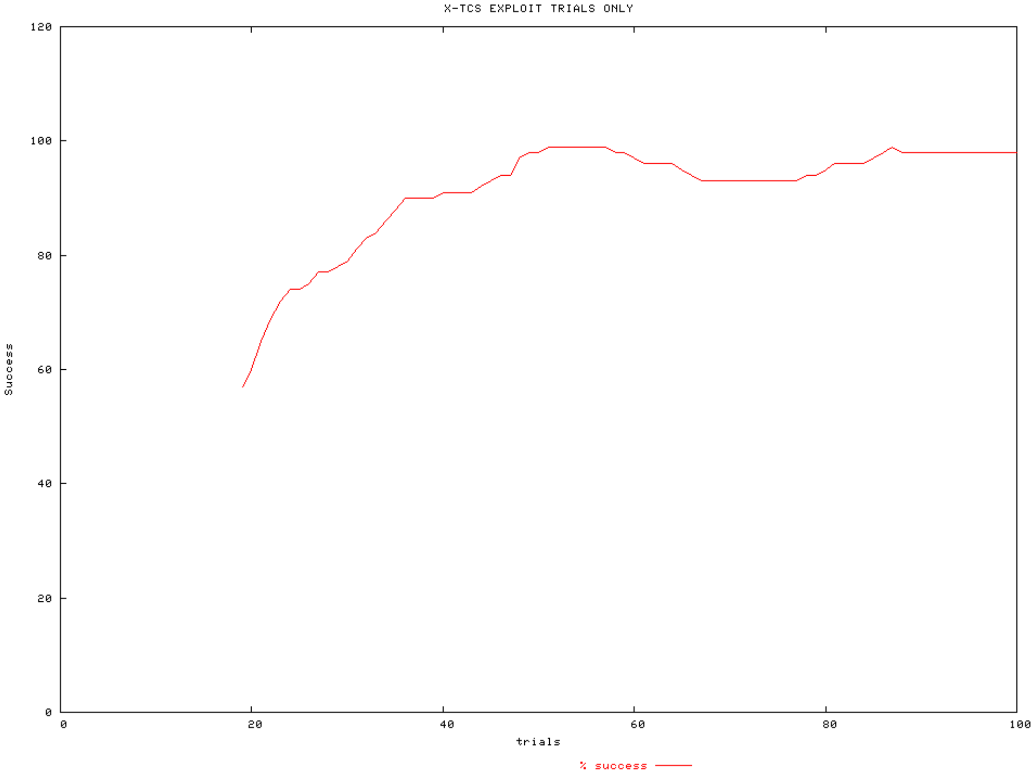


Figure 5-11 Windowed average % success. Exploit only

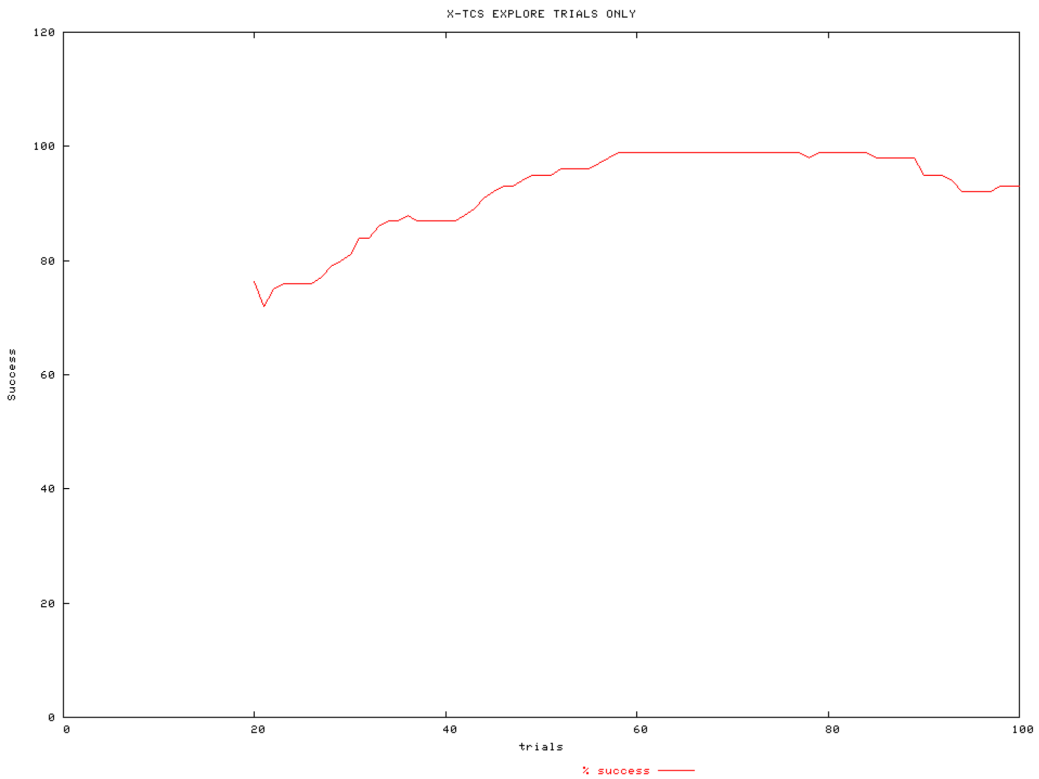


Figure 5-12 Windowed average % success. Explore only

Figure 5-13 and Figure 5-14 show windowed average time to goal and windowed average percentage success for the exploit and explore trials of a single run, as do Figure 5-15 and Figure 5-16 for a different run. In the first of these pairs of graphs, X-TCS shows itself to be capable of almost perfect performance. In the exploit trials, the robot consistently achieves an average time to goal which is within the optimal bounds, while in the explore trials average time to goal is only slightly worse. In both cases, percentage success rises rapidly to the optimum, deviating only occasionally. In Figure 5-15 and Figure 5-16 a second individual run is shown. In this case performance was slightly more variable. Apart from the differences in the initial population which might account for variation, each trial starts with the robot at only approximately the same position and orientation in the environment, and there may be delays in response from the power supplies thereby introducing variation into the signalling of reward.

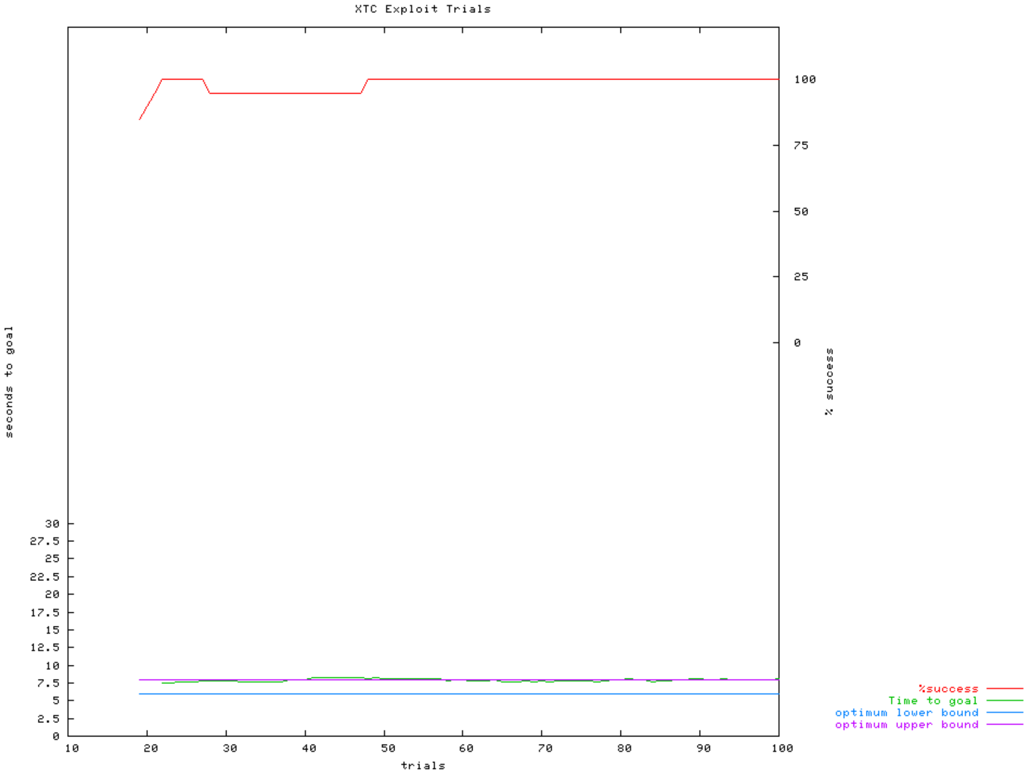


Figure 5-13 Individual Trial 1. Windowed average % success and time to goal. Exploit only

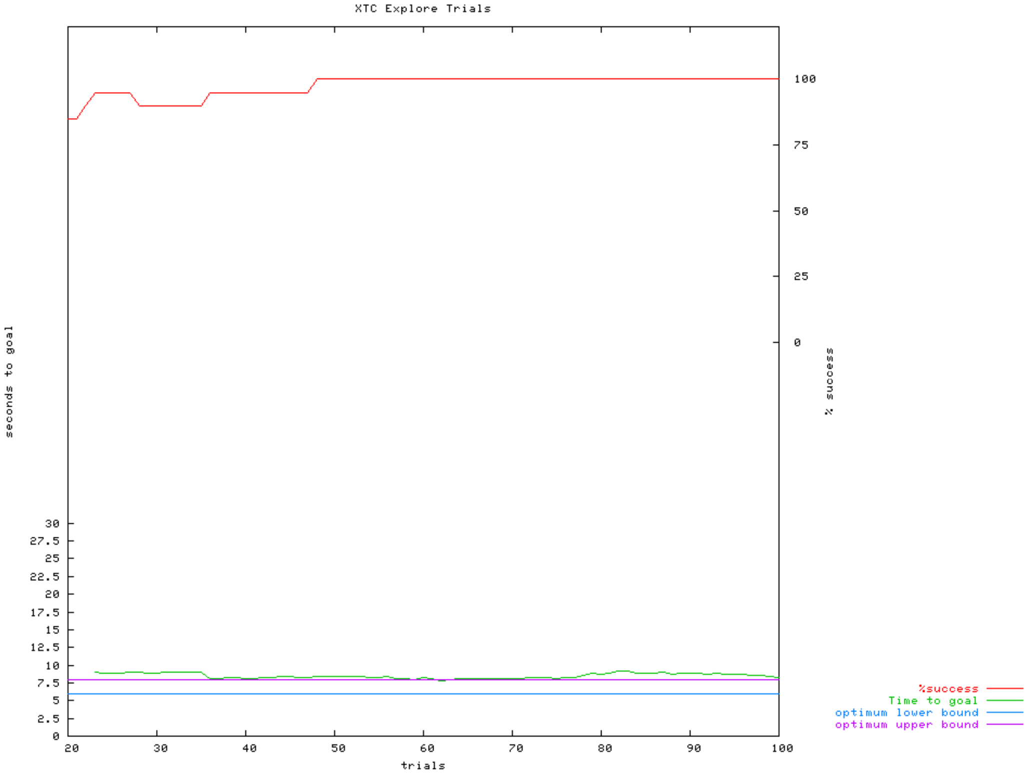


Figure 5-14 Individual Trial 1. Windowed average % success and time to goal. Explore only

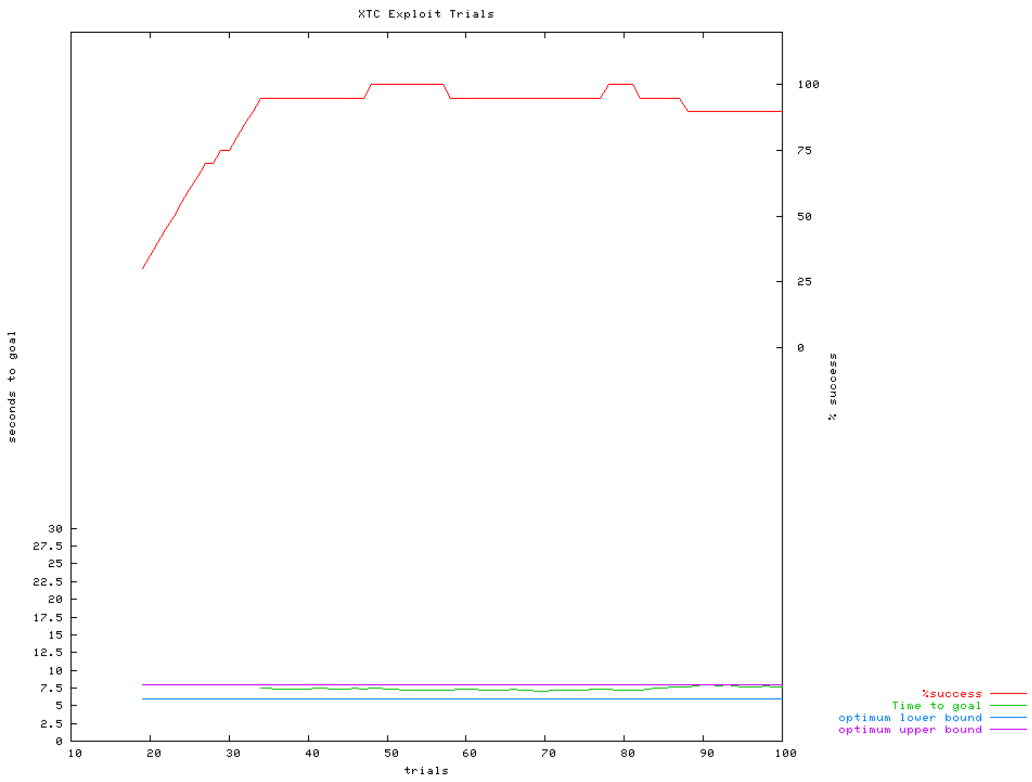


Figure 5-15 Individual Trial 2. Windowed average % success and time to goal. Exploit only

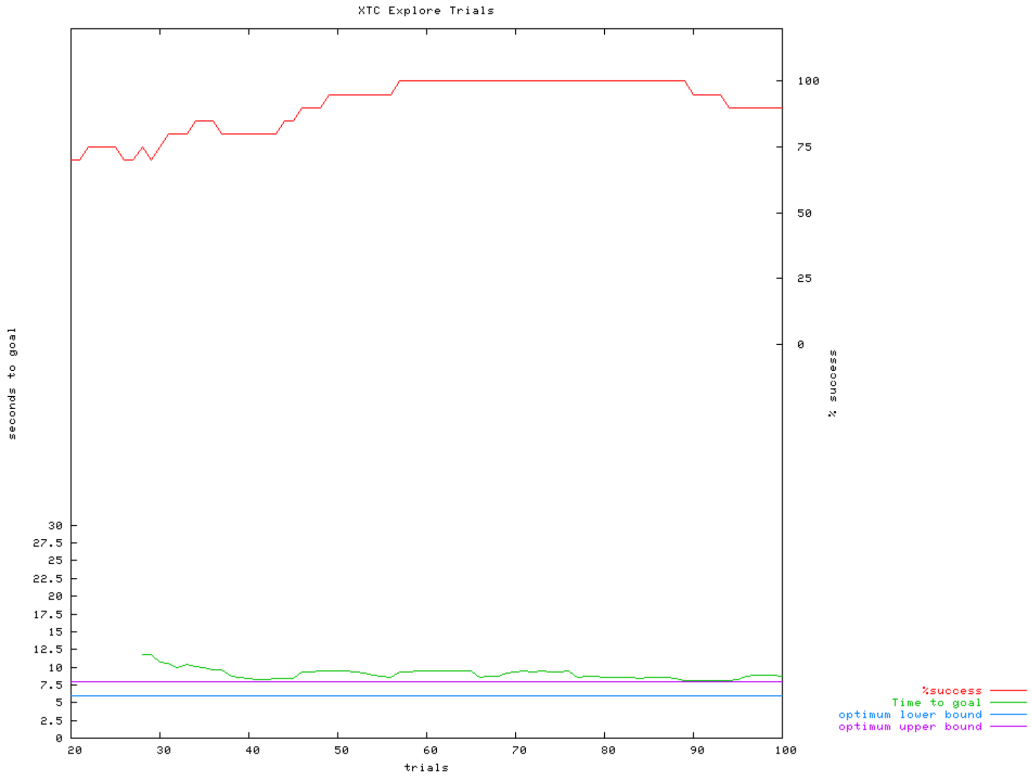


Figure 5-16 Individual Trial 2. Windowed average % success and time to goal. Explore only

In comparison with TCS, the behaviour of the robot seemed strikingly better. After a short initial phase the robot quickly learned to home in on the goal, and having learned this deviated seldom from the desired behaviour. This subjective impression is validated by the statistics in Table 5-1. When the overall average of the failure to success ratios of X-TCS and TCS are compared, we see that X-TCS has a lower mean. Thus X-TCS more reliably achieves success in this single objective task than TCS (statistically significant at the 95% confidence level).

X-TCS (average of both explore and exploit)	0.155 ± 0.033 (Mean ± SE. n=100)
TCS	0.294 ± 0.038 (Mean ± SE. n=200)
t Stat 2.737, t Critical two-tail 1.968	

Table 5-1 Overall average failure to success

Before embarking on these experiments with X-TCS, it was uncertain how XCS would fare. XCS depends for its success upon building a complete map of the payoffs of taking all possible actions. It seemed possible that the robot would take many small actions as it explored the space, resulting in inefficient locomotion. Subjectively, it appeared that X-TCS did take more small actions than TCS, although the trajectories the robot followed were much less snakelike than the winding paths of alternating left and right turns that TCS seemed often to adopt. Under the control of X-TCS, the robot appeared to turn to orientate itself and then drive forwards giving the subjective impression of more purposeful behaviour than the waggling progression achieved with TCS.

Figure 5-17 shows the average number of actions taken by TCS in the one-objective task reported in Section 3.5.1, compared with the number of actions taken in the experiment described above. With a two-tailed t-test, there seems no reason to reject the null hypothesis that there is no difference between the means, as shown in Table 5-2.

t-Test: Two-Sample Assuming Unequal Variances
Number of actions taken to reach the goal.

	TCS	*X-TCS*
Mean	11.27723	12.35402
Variance	35.06738	44.72051
Observations	100	100
Hypothesized Mean Difference	0	
df	197	
t Stat	-1.2115	
P(T<=t) two-tail	0.227156	
t Critical two-tail	1.97208	

Table 5-2 Two-Tailed t-test. 100 trials TCS, 100 X-TCS

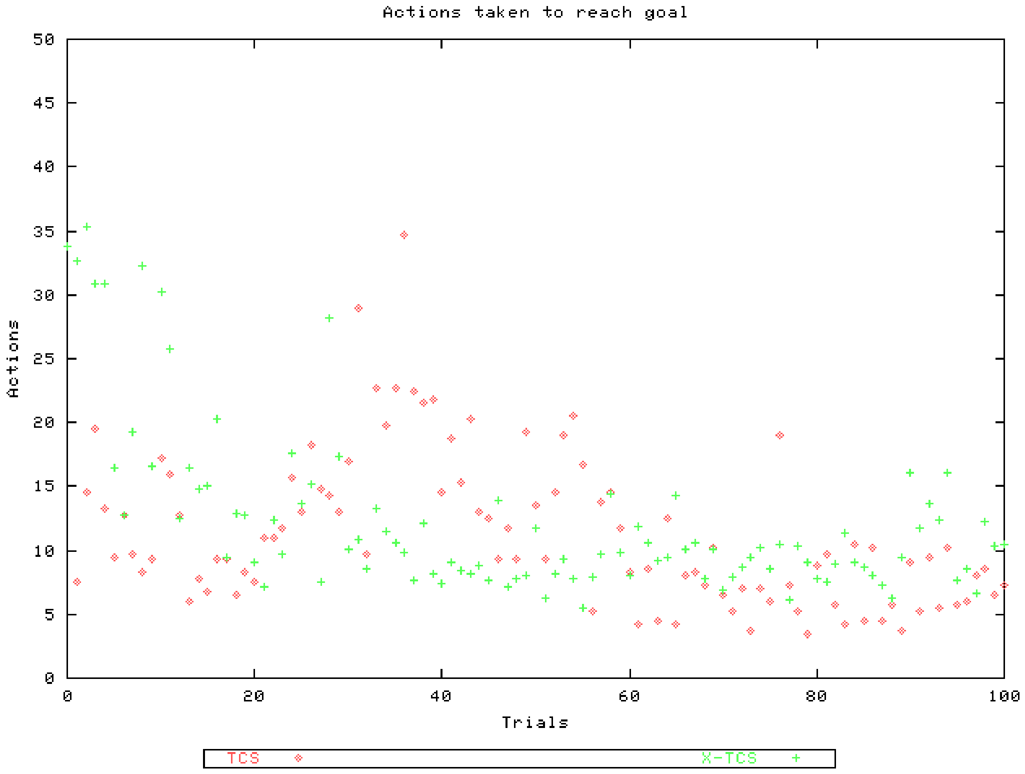


Figure 5-17 Comparison of X-TCS and TCS actions per trial

5.4.2 Two Objectives

Having shown that X-TCS is capable of achieving near-optimal behaviour in the simple one objective task, results in the two objective task are now presented. The experimental set-up remains as detailed in Section 3.5.2.

X-TCS here performs an explore and an exploit trial with an internal energy level of less than 0.5, and then both an explore and exploit trial with an internal energy level of greater than 0.5. The figures that follow show both the explore and exploit trials, only exploit, and only explore, and show metrics for both goal states. All graphs are the average of five experiments. An experiment lasted 100 trials, taking around three and a quarter hours to complete. N=2000, σ is 0.05 and η is 0.1. These results are therefore directly comparable to those shown in Section 3.5.2, in which TCS was not shown to be capable of achieving optimal performance, even when the learning process was allowed to continue for 1200 trials.

In Figure 5-18, Figure 5-19 and Figure 5-20 the number of actions taken to reach the goal is shown. Again, it can be seen that there is a tendency for the number of actions taken to reduce as learning continues.

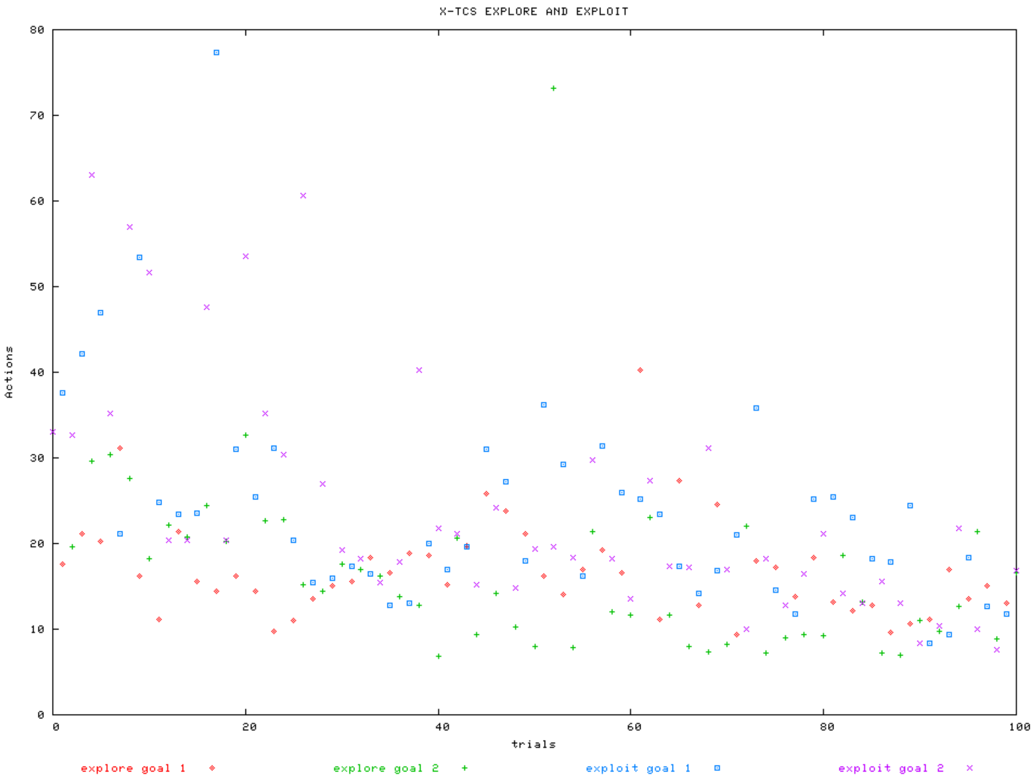


Figure 5-18 Two Objectives. Actions taken. Average of Explore and Exploit trials.

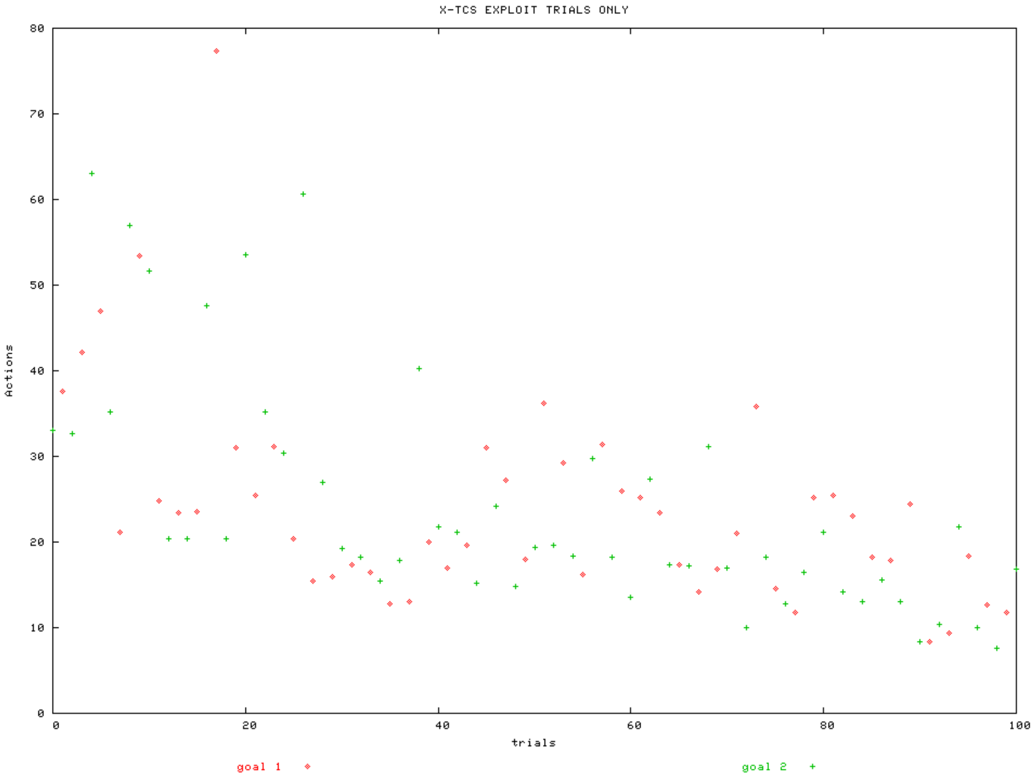


Figure 5-19 Two Objectives. Actions taken. Exploit only

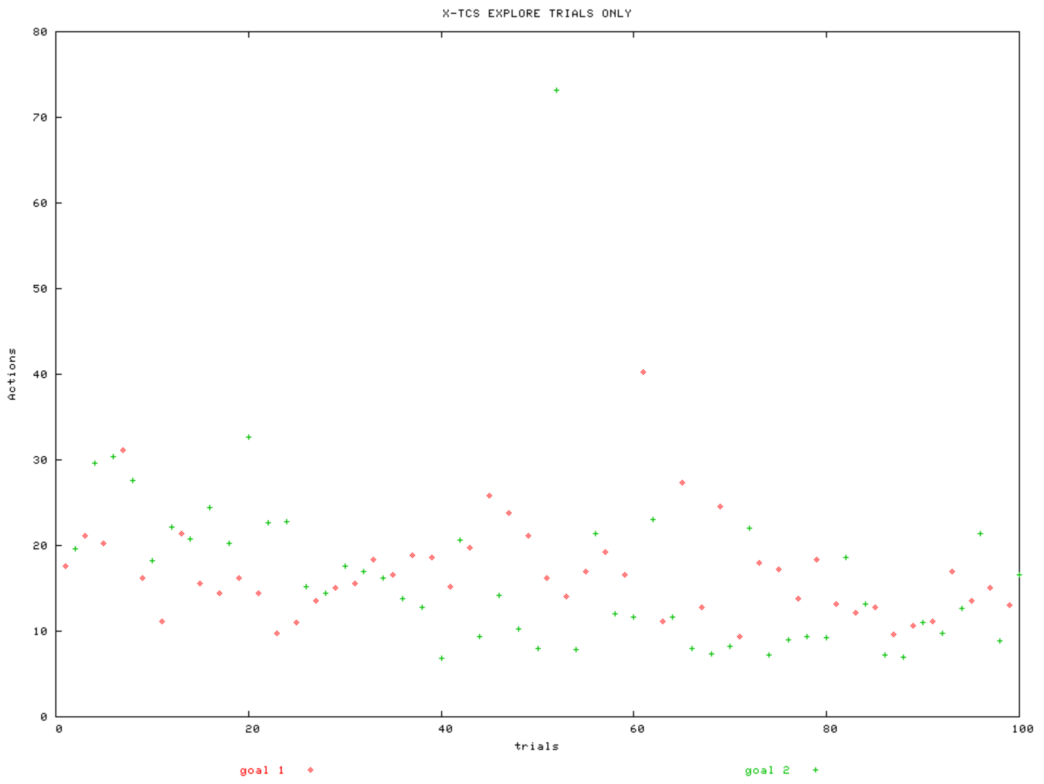


Figure 5-20 Two Objectives. Actions taken. Explore only

Figure 5-21, Figure 5-22 and Figure 5-23 show the ratio of failure to success. As was the case with TCS, a trial is counted as a failure if the goal reached is not the correct one or if the trial continues for more than 30 seconds. As in the single objective task, the ratio of failures to success decreases with successive trials. In both explore and exploit trials the problem is quickly learned and thereafter there is a small and decreasing tendency to occasionally fail.

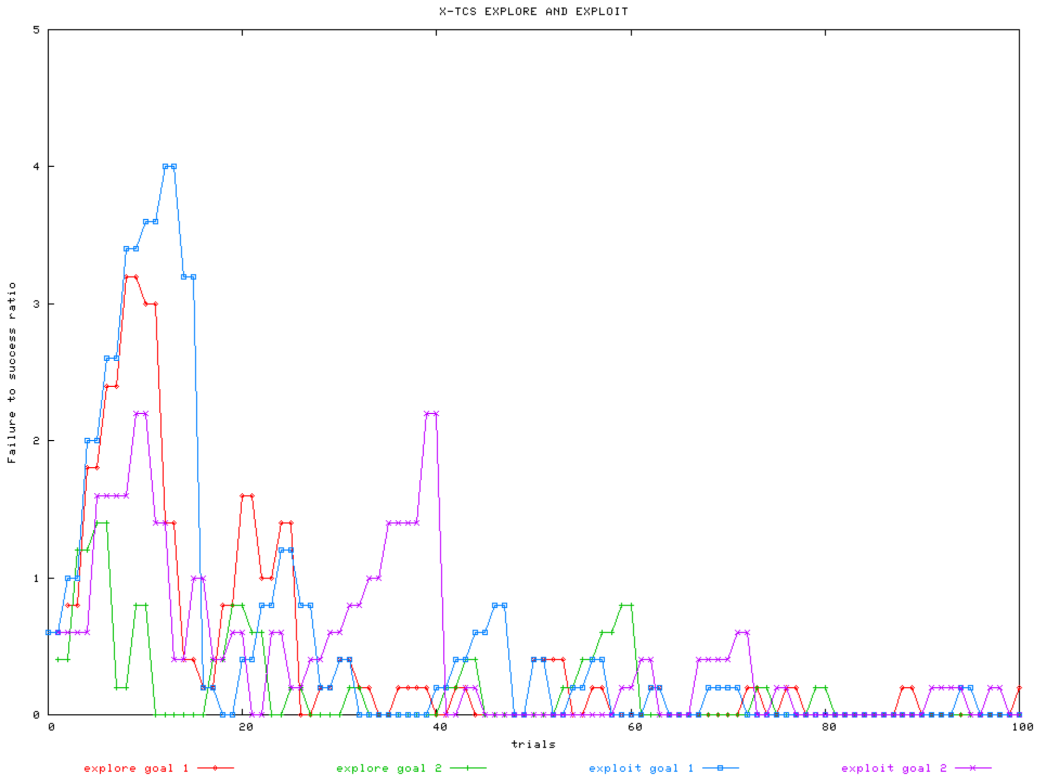


Figure 5-21 Two Objectives. Failure to Success Ratio. Average of Explore and Exploit.

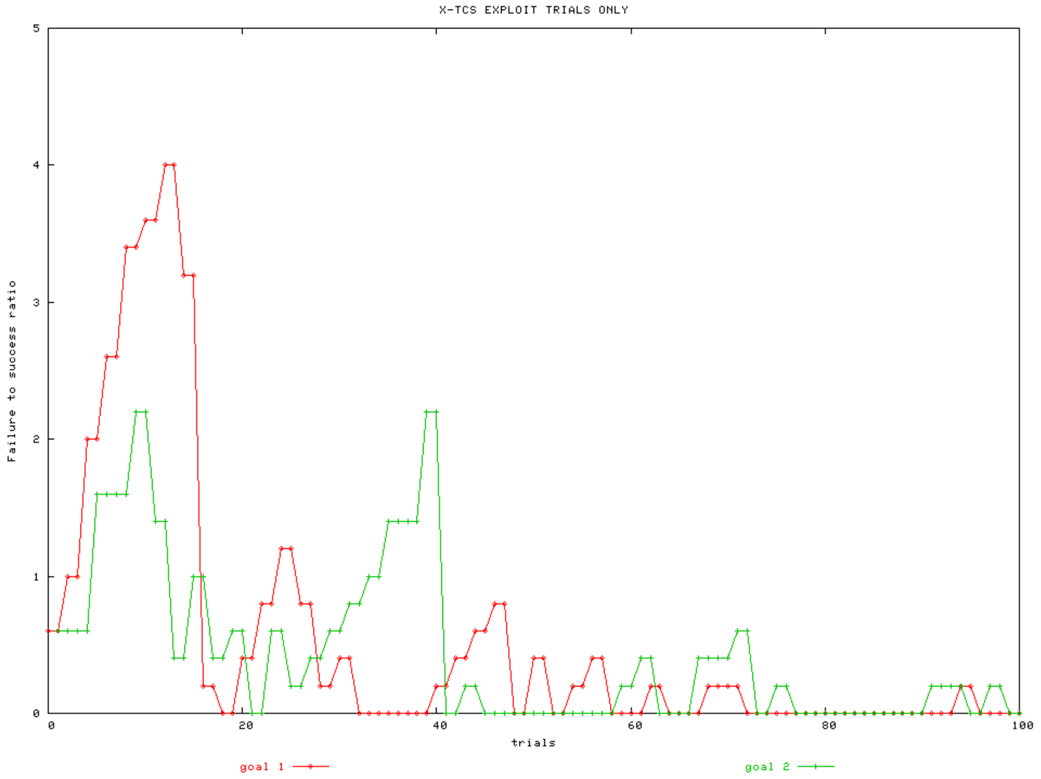


Figure 5-22 Two Objectives. Failure to Success Ratio. Exploit only

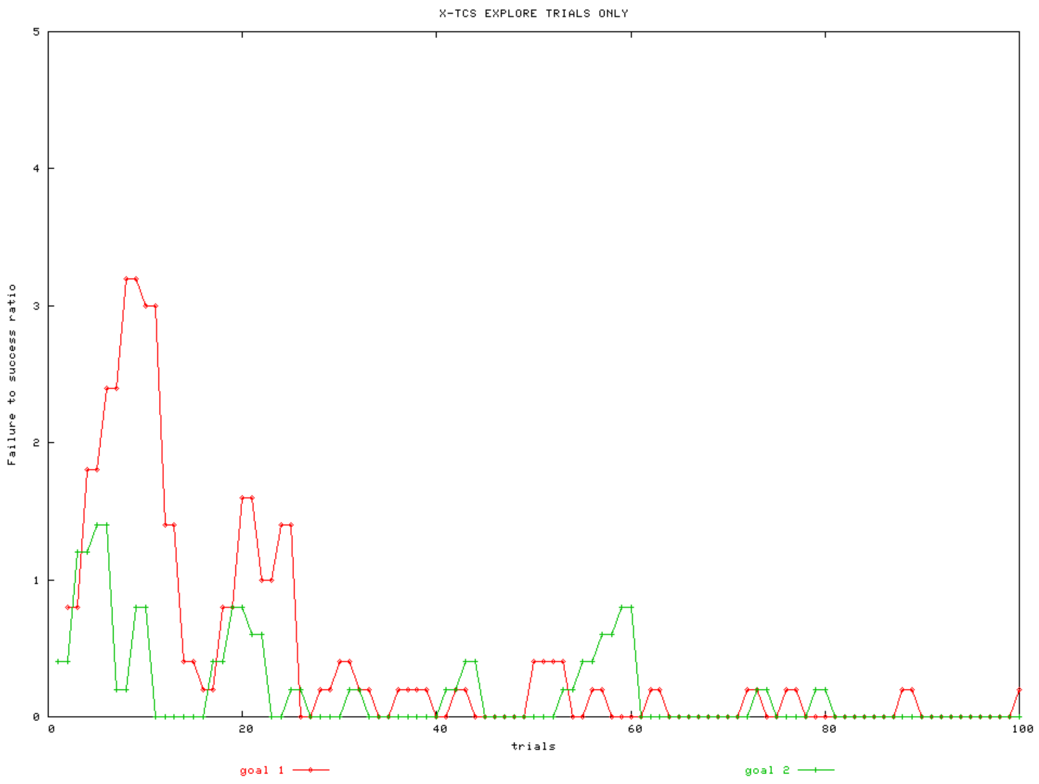


Figure 5-23 Two Objectives. Failure to Success Ratio. Explore only

Figure 5-24, Figure 5-25 and Figure 5-26 show the amount of time taken for the robot to reach its goal. The amount of time taken decreases as learning continues, approaching the upper limit of the optimal band. However, the robot generally takes longer to achieve its goal in the two objective problem than in the single objective case. The spread of the results also decreases as learning continues, showing that the system is becoming more consistent.

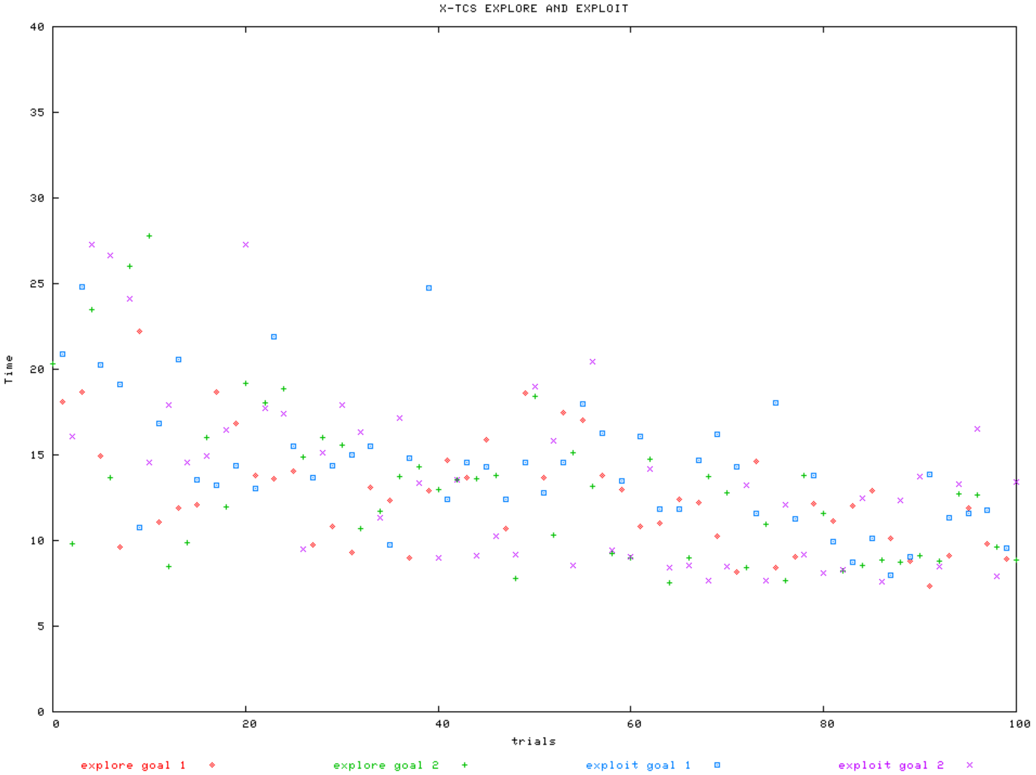


Figure 5-24 Two Objectives. Time to Goal. Average Explore and Exploit

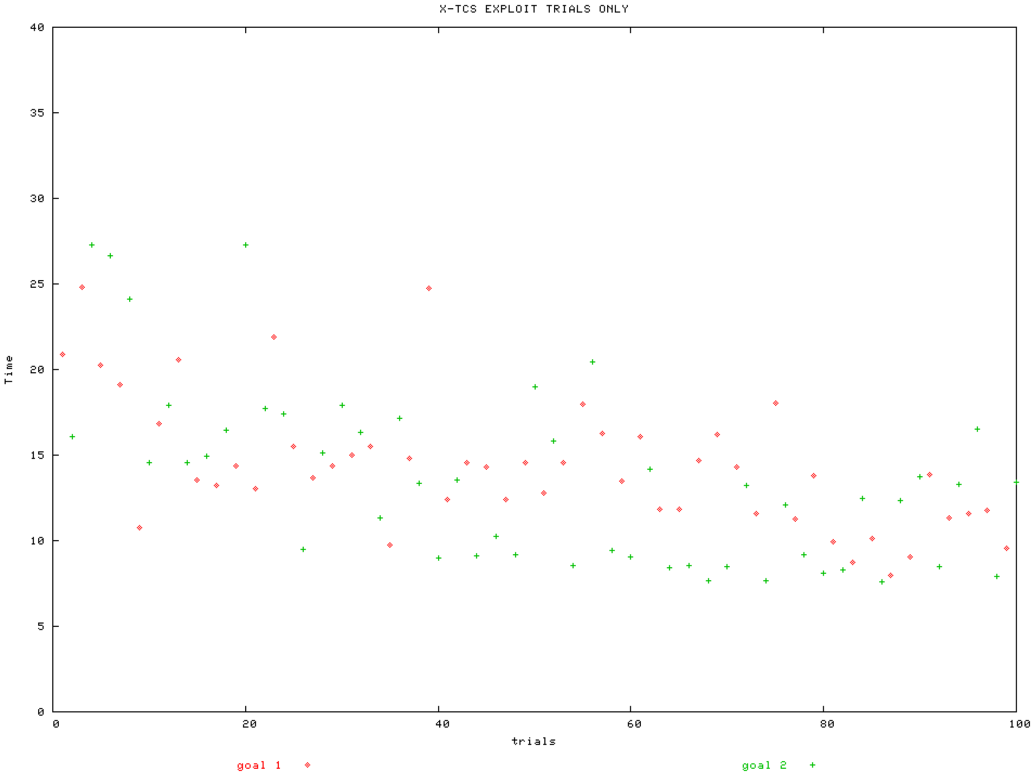


Figure 5-25 Two Objectives. Time to Goal. Exploit only

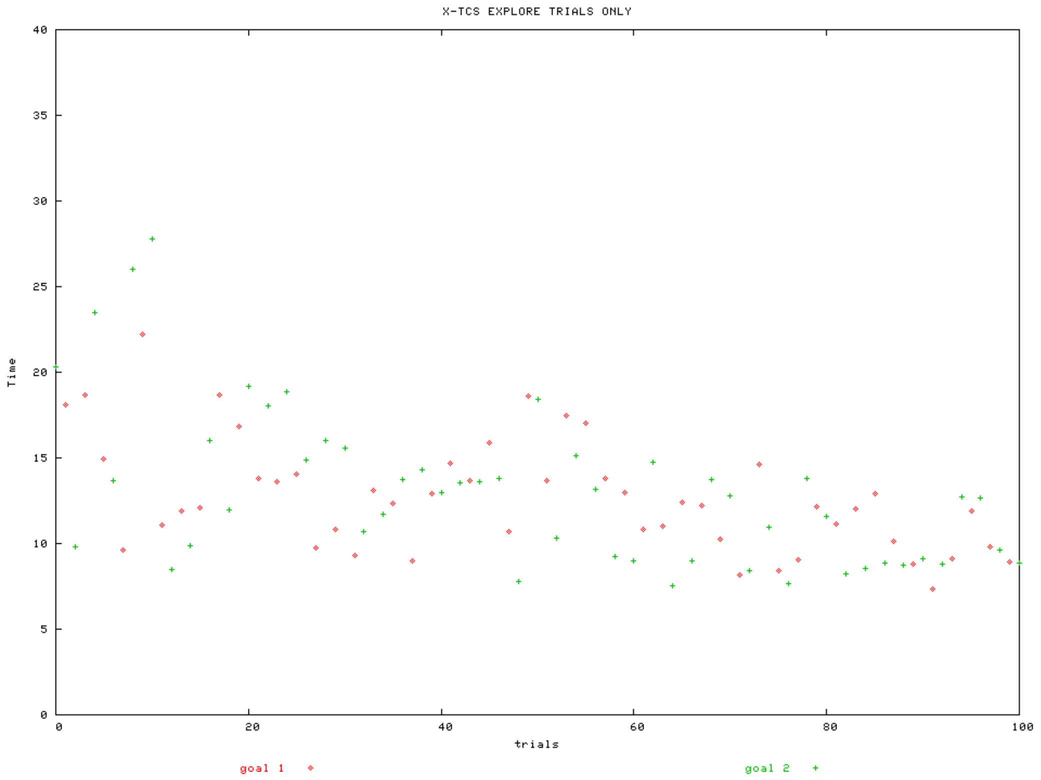


Figure 5-26 Two Objectives. Time to Goal. Explore only

Figure 5-27, Figure 5-28 and Figure 5-29 show the windowed average of the percentage successful trials. In contrast to the results presented for TCS in the comparable Figures 3-11 and 3-12, we see a much improved picture. Where TCS varied considerably between individual runs and showed little sign of approaching 100% success, the performance of X-TCS swiftly improves and approaches the optimum after only 100 explore and exploit trials.

This shows a marked improvement over TCS for the dual objective problem (see Figure 3-10). Figure 5-27 which shows the average performance of both explore and exploit trials shows that X-TCS learns quicker than TCS, and achieves better results.

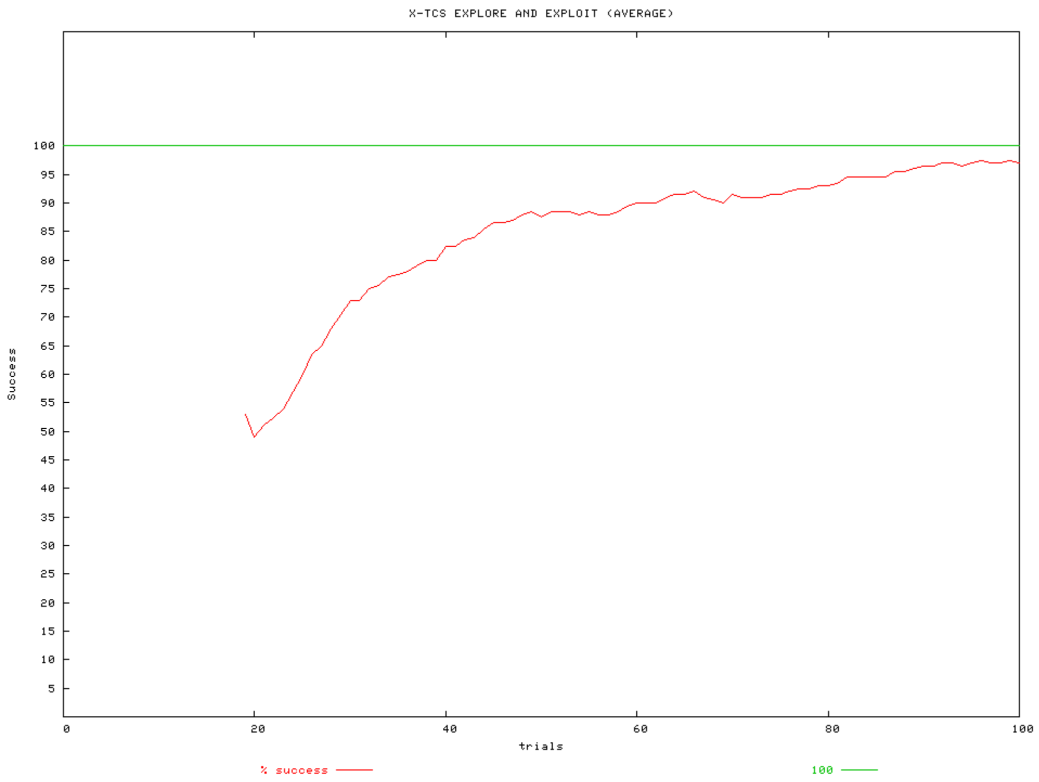


Figure 5-27 Two Objectives. Windowed average percentage success. Average Explore and Exploit

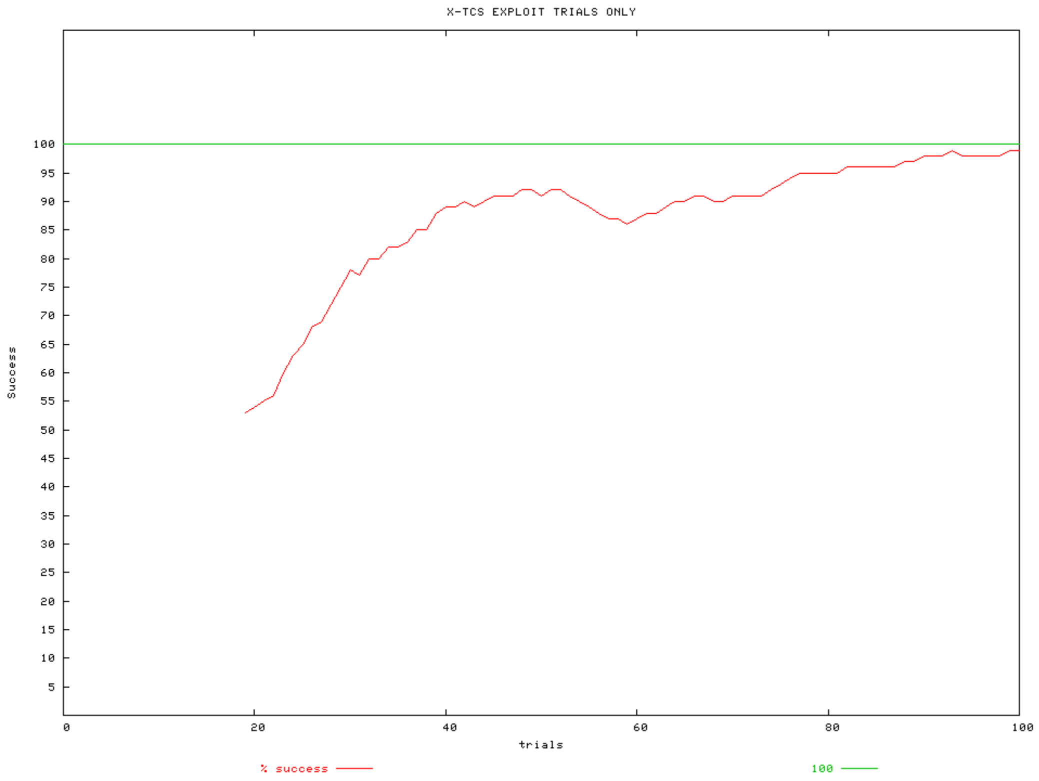


Figure 5-28 Two Objectives. Windowed average percentage success. Exploit only

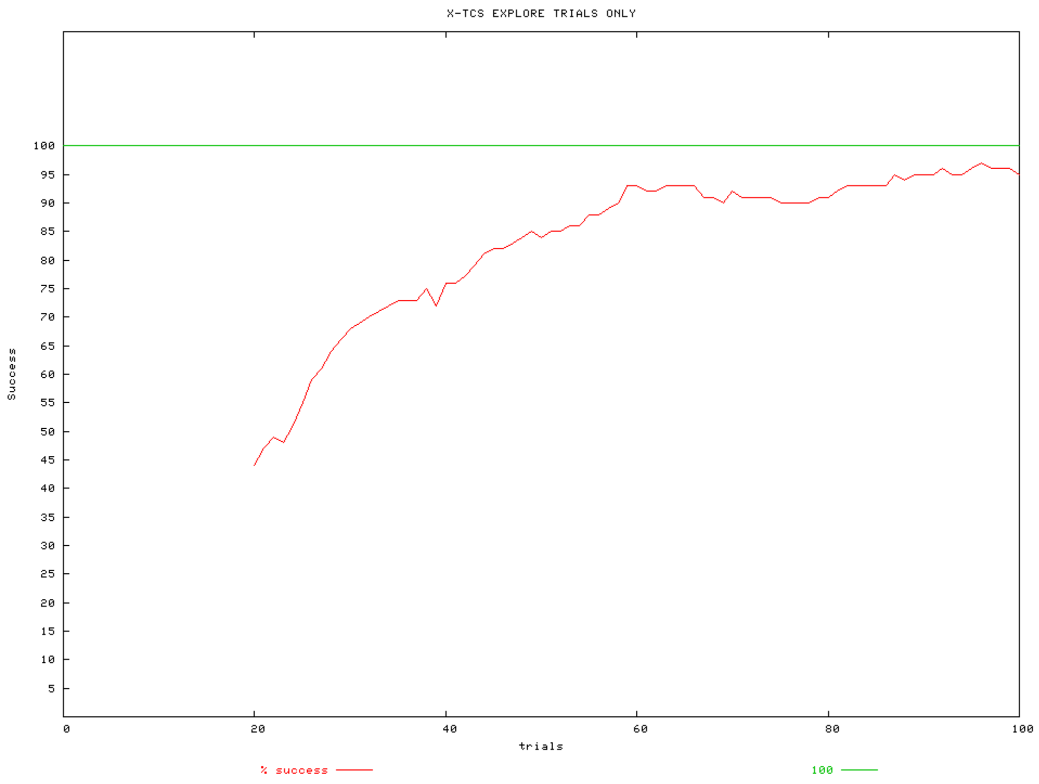


Figure 5-29 Two Objectives. Windowed average percentage success. Explore only

Figure 5-30 and Figure 5-31 show results for the best of the five trials which produced the average results presented above. Figure 5-32 and Figure 5-33 show the worst of the trials. In Figure 5-30 and Figure 5-31 we see that percentage success rises swiftly until it attains 100%. After achieving this, the learner does not deviate, consistently staying perfect in respect to attaining the correct goal as determined by its internal energy level. Perfection is attained quicker in the exploit trials than in the explore trials. Since there is no deviation from perfection in the explore trials it is clear that accurate rules have been discovered and have come to dominate such that roulette wheel selection has in effect self-annealed, and become deterministic. Time taken to achieve the goal state is less optimal. In Figure 5-30 it can be seen that time to goal is suboptimal until approximately trial 80, after which it attains the upper limit of optimality. In contrast the explore trials are less good with respect to time taken, and deviate away from optimality slightly after achieving it. This suggests that exploration is continuing after all.

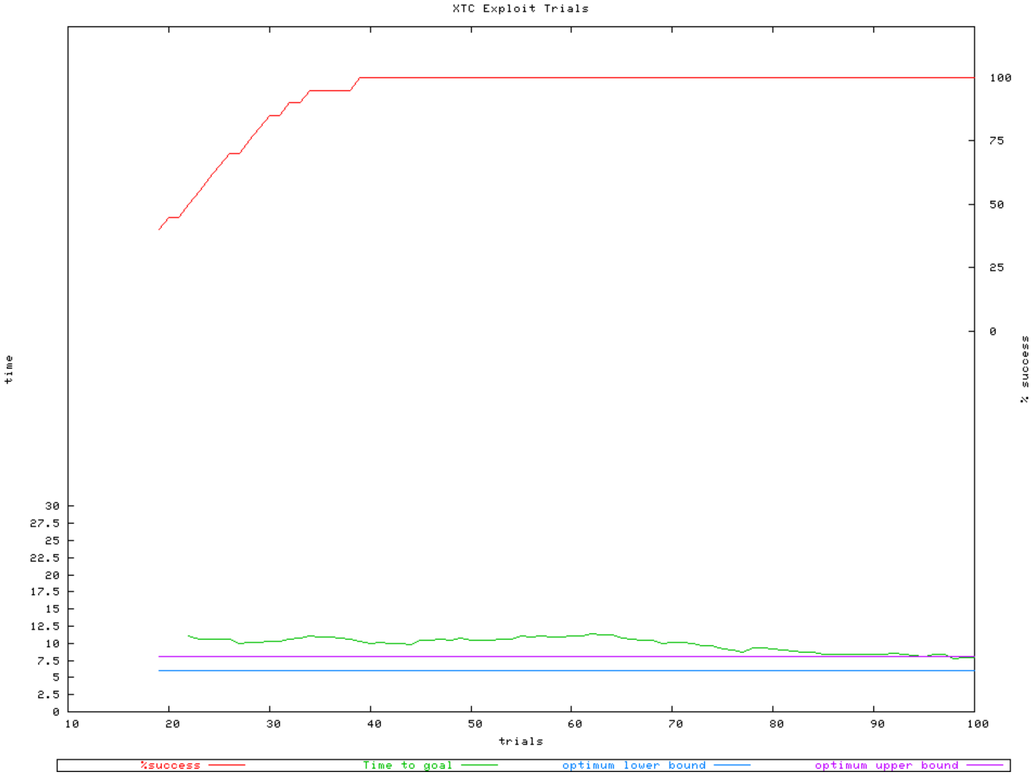


Figure 5-30 Two Objectives. Run 1. Exploit only

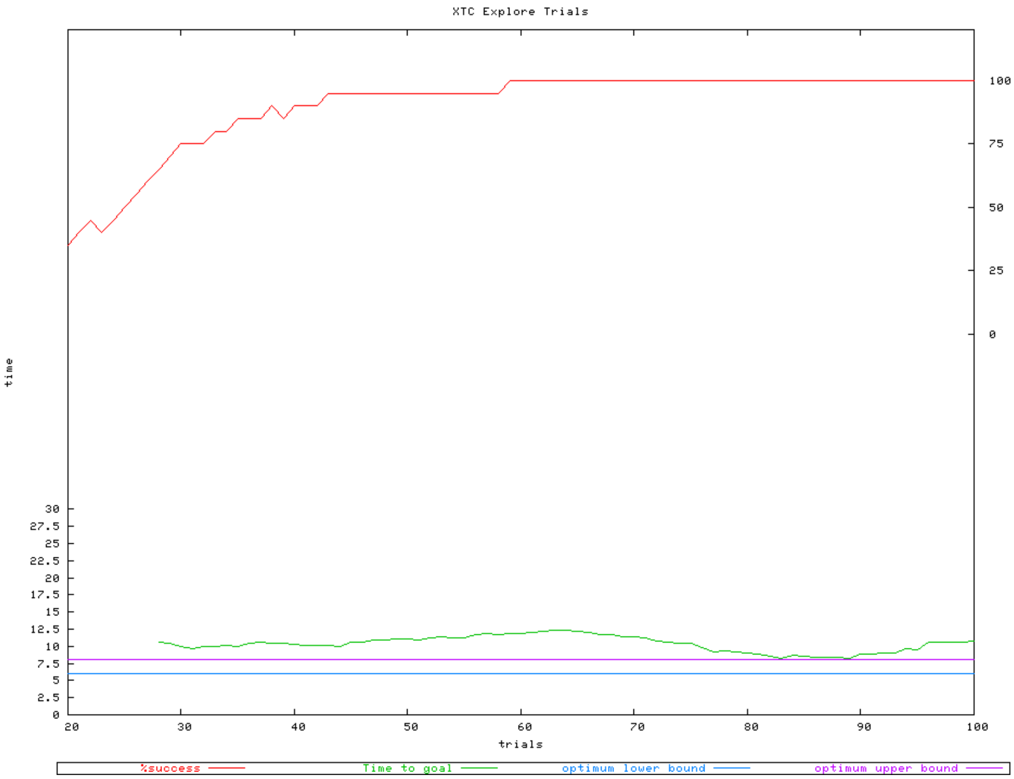


Figure 5-31 Two Objectives. Run 1. Explore only

Figure 5-32 and Figure 5-33 in contrast show the worst of the individual runs. In the first of these we see that in the exploit trials learning is much slower, but again, eventually converges upon 100% success. Time taken is worse, showing little sign of converging to the optimal band. In the explore trials there is a slow increase in percentage success, finally approaching 100%. Time taken is worse still, although it seems to be improving towards the end of the experiment.

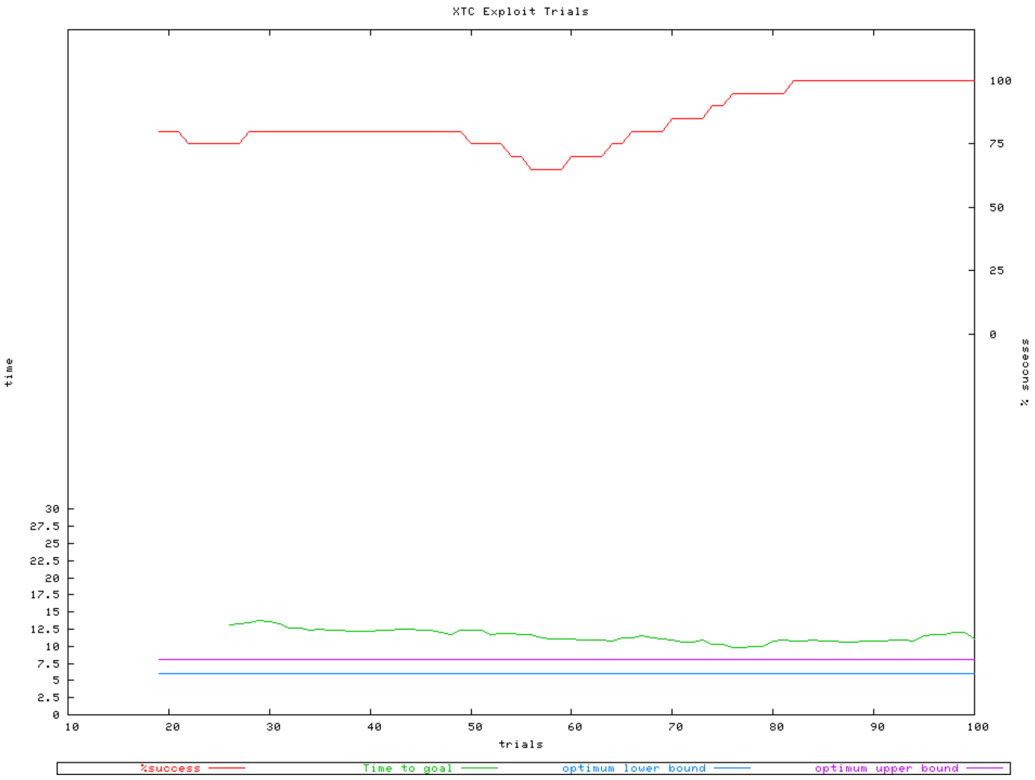


Figure 5-32 Two Objectives. Run 2. Exploit only

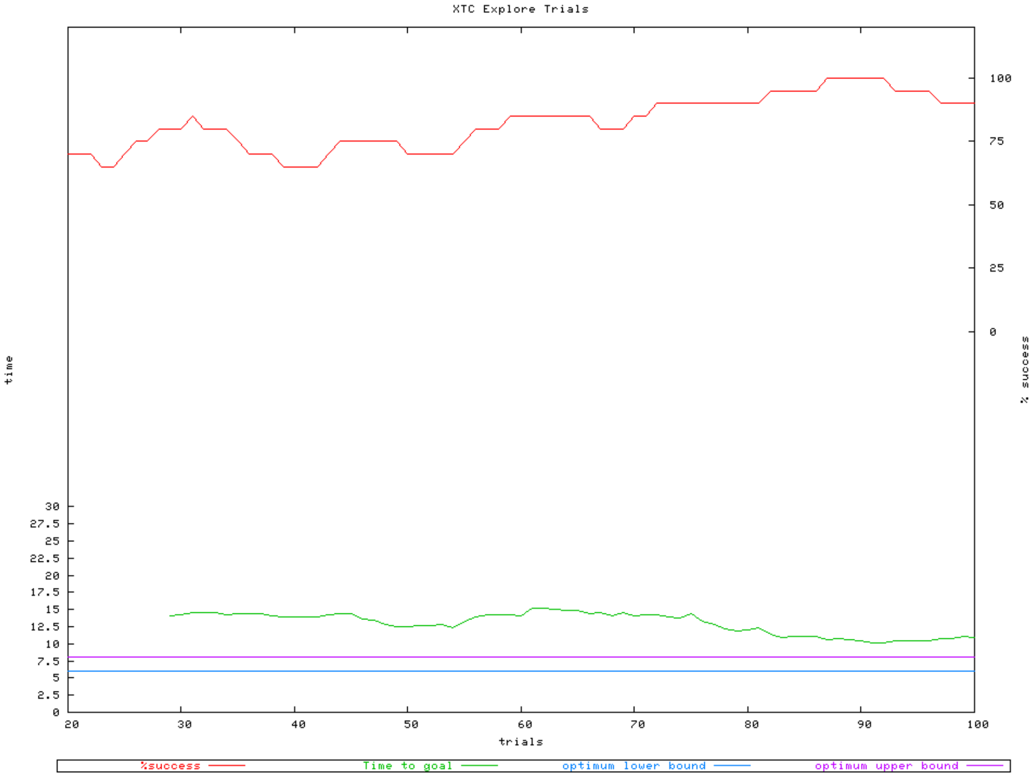


Figure 5-33 Two Objectives. Run 2. Explore only

5.5 Conclusions

In this chapter have been described the alterations to XCS which are necessary to incorporate the changes Hurst made to ZCS, thereby producing TCS. The new system has been named X-TCS. In simulation we have seen that XCS requires far less parameter adjustment than ZCS in order to achieve optimal performance. We saw previously that TCS was unable to solve optimally the two objective robot task, even when run for very long periods of time.

In contrast, X-TCS has shown the same ability to optimally solve the two objective problem on the robot that XCS showed in simulation. X-TCS seems good when assessed against the desirable characteristics for a robot learning system outlined in Section 3.2.3 . The online behaviour (i.e. the average of both explore and exploit trials) of the robot approaches 100% performance, which would certainly not have been the case had a random exploration policy been used. Optimality was achieved after a relatively short period of time; X-TCS is quick to learn. In comparison to the purely evolutionary robotics approaches discussed in e.g. [Floreano1996] where a controller for a single objective navigation task was evolved after ten days, X-TCS achieved optimal behaviour on the similar single objective task detailed here after an average of about two and a half hours, which included the time taken to restore the robot to its starting state by a random walk around the arena. Even the worst individual experiment presented for the dual objective task eventually approaches 100% success, and the average behaviour of the learner is very nearly perfect. In contrast to ZCS and – it seems likely – TCS, there was no necessity for careful adjustment of the reinforcement learning parameters in order to achieve these results, and the parameters were indeed set the same as used in the TCS experiments. Although X-TCS takes more actions to attain its goals than TCS, this difference is small. Even though X-TCS takes more actions, its performance is more nearly optimal than that of TCS; the larger number of actions indicates a more specific solution, suggesting that TCS may have suffered from over-generalization.

In conclusion, X-TCS seems to satisfy the criteria for a good learning algorithm for simple robot control problems in which the robot's behaviour must alter according to the internal state of the learning system. Learning is fast and attains near optimal results with no necessity for manual intervention. There was no *a priori* discretisation imposed, no assumptions made about how the problem would be solved, and no need for parameter adjustment.

Chapter 6 Conclusions and Future Work

This chapter briefly reviews the work presented in this thesis, and then suggest some avenues of investigation which might be profitable for future research.

6.1 Conclusions

The focus in this thesis has been upon the practicalities of learning on a robotic platform. It was asserted that, in any but the most trivial applications, a robot will need to switch between the satisfaction of different goals depending on its internal state. The use of various LCS was examined for problems of robot control with multiple goals in both simulated and physical environments.

We saw that ZCS was capable of optimal performance in simple dual objective tasks in simulation. However, this was only true when its parameters were set correctly; a difficult task since some parameters cannot be tuned in isolation. In order to set the reinforcement learning parameters of ZCS the performance of the system was examined – measured in terms of both the steps taken in a trial, and the percentage of trials in which the goal state achieved was the optimum one given the animat's internal state – as it varied with different settings of the RL parameters. To the author's knowledge, these parameters have otherwise been set by trial and error by the experimenter, as in [Bull2002a] in which ZCS was first reported to be capable of optimal behaviour. Using a process of refinement it was possible to find parameter sets that allowed ZCS to perform optimally in the simple dual objective tasks. However, it was not possible to find a parameter set that would allow ZCS to solve optimally the slightly more complicated tasks such as ‘woods1e-type3’. Similarly, it was not possible to find parameter sets allowing the optimal solution of a simple three objective problem.

Given the need to carefully select the parameter set in order to enable ZCS to solve even these simple problems, it seemed an unlikely candidate as a practical means of enabling a robotic learner to solve problems with multiple goals. Although Bull et al. showed the successful self-adaptation of some parameters [Bull2000a, Bull2000b], they were unable to achieve this with the RL parameters without modifying the algorithm to enforce co-operation between rules in successive action sets, and even so were unable to achieve optimality. Using a meta-heuristic such as an evolutionary algorithm to search the space of parameter combinations automatically would be possible, but likely to take an unacceptable amount of time.

Tthe Temporal Classifier System TCS was implemented on a robotic platform, and achieved results in a single objective task similar to those reported by Hurst et al. However, it was unable to achieve reliably optimal performance with TCS in the dual objective task. To gather enough results to draw statistically valid conclusions took many hours; days with experiments comprising a thousand trials. Since ZCS is so dependent on the settings of its parameters, it seemed likely that TCS is equally sensitive. This suggested that TCS was not a suitable candidate to solve the problem of a practical robotic learner for problems with multiple goals.

In the same simulated problems solved by ZCS, XCS showed itself equally able to achieve optimal performance. However, XCS required much less tuning of its parameters, making it a more practical choice. XCS was able to solve problems that had proved intractable to optimal solution with ZCS. Typically, XCS is implemented with alternating deterministic action selection ‘exploit’ trials, and random action selection in the ‘explore’ trials. Any robot that acts randomly half the time is likely to be of little use. This motivated an exploration of the use of a roulette-wheel action selection policy in the explore trials. This was shown to be capable of achieving optimal results. Interestingly, it was found that roulette-wheel selection required a larger population than random action selection to achieve fully optimal performance, but that at population sizes where random exploration was itself sub-optimal, could offer a performance advantage. We noted that this might be advantageous on a robotic platform where resources may be limited and explored ways in which the action selection policy was linked to performance at different population sizes, thus linking this work to the emerging theories of Butz et al.

X-TCS was then implemented and experiments on the robotic platform were described. This algorithm implements the features of TCS – the automatic determination of what constitutes an environmental event, and a temporally-adjusted reinforcement schedule – within the framework of XCS. X-TCS showed itself to be capable of achieving optimal results on the single objective problem. In the dual objective problem, with the same settings of population size, and reinforcement parameters as TCS, X-TCS quickly demonstrated nearly optimal behaviour.

X-TCS required little *a priori* knowledge or problem-specific adjustment in order to achieve these results. There was no necessity to discover elusive settings of the many parameters, neither manually nor by employing a heuristic. There was no need to impose an *a priori* scheme in which the world is rendered into a grid of states of predetermined size. Learning was reliable and rapid. In most experiments lasting just over three hours for 100 trials, the robot was reliably achieving 100% accurate performance after around 20 trials; around 40 minutes. Of this, an average of 22.6 minutes was actually spent in learning – the rest of the time was spent returning the robot to its starting position by a random walk around its arena.

With no need to tune the algorithm, rapid real-time learning, reliably near-optimal performance, and without the need for the experimenter to tailor the algorithm to the problem or to build models, X-TCS seems to satisfy the criteria listed on page 110 of being a truly practical mechanism for learning to solve these simple problems with multiple goals on a robotic platform.

6.2 Future Work

The problems presented in this thesis have been simple, in order to prove the utility of different approaches. Future work should improve utility, and explore more complicated problems.

6.2.1 Self Adaptation of Parameters

Hurst explored the use of self-adaptation of mutation rate and learning rate in XCS [Hurst2003]. He demonstrated that the use of a self-adaptive mutation rate improved the ability of XCS to solve multi-step problems in which long chains of actions are needed before reward is given. This could clearly be of benefit in X-TCS. In particular, he shows that in replicating experiments performed by Lanzi [Lanzi1999] the use of self-adaptive mutation greatly improves the otherwise disappointing performance he observed while using roulette-wheel selection in Woods 12-14.

Hurst (ibid.) suggests that *'it is difficult to adapt the learning rate in XCS'*, and that *'self-adaptation of β learning rate within ZCS was a failure'*, citing a lack of selective pressure for the latter. In the work of Butz et al. incorporating gradient descent into XCS, they suggest that since they adjust the update procedure for a classifiers prediction according to the ratio of its fitness to the sum of fitnesses of others in the same action set, this represents an adaptive learning rate [Butz2004]. They show an increase in performance in long multi-step environments such as Woods14 which XCS is generally believed to be unable to solve without the use of *specify* or *teletransportation* [Lanzi1999] .

Since these adaptive techniques allow XCS to solve problems with longer chains of actions, they could be useful avenues for investigation in order to apply X-TCS to more complex real-world problems. There may also be great benefit in further investigating the use of self-adaptive parameters in TCS, since parameterisation was believed to be the cause of its sub-optimal behaviour in the dual objective robot task.

The self-adaptation of parameters might also hold the key for the successful and general application of TCS.

6.2.2 Action Selection Policies

XCS, as used here in its original form and as the basis for X-TCS, uses fitness proportionate selection in its GA. Butz et al. [Butz2003c] suggest that the use of tournament selection offers a number of advantages. Firstly, it may aid the production of accurate classifiers that more quickly replace overgeneral rules. Secondly, it may make the system more resilient in noisy environments. Thirdly, it may promote good performance with less dependence on parameter settings. They show an improvement in performance in the Woods6 multi-step environment which proves insoluble in the absence of tournament selection.

Butz et al. find no problem where performance suffers from the inclusion of tournament selection, and many that benefit. Since there may be a benefit in noisy environments such as experienced with a physical robot in the real world, and also since there may be further benefits in terms of stability in respect to parameter settings, tournament selection would seem an interesting technique to include in X-TCS. We might expect it to speed up learning, allow the use in more noisy environments, and reduce still further the necessity to tune parameters to the problem at hand. However, recent work by Kharbat et al. [Kharbat2005] suggests that tournament and roulette-wheel selection are both equally dependent upon parameter settings and may be equivalent in terms of performance. They also find that rule compaction is more efficient with roulette selection due to its inherent bias towards the fitter individuals in a population.

6.2.3 Noise

Although no special precautions were taken to insulate the robot experiments from the variations in light occasioned by diurnal variation, the British climate and passing students and visitors, we saw good performance. We must assume that the levels of noise present in the environment did not prove a significant challenge to X-TCS in this setting.

Christopher Stone of the UWE reports[16] that in single-step problems in simulation in which varying amounts of Gaussian noise are introduced, XCS performs less well than ZCS as noise levels are increased, until it is finally unable to learn at all.

The noise in these experiments may be quite different from that experienced in Stone's experiments. Firstly, Stone is applying constant noise by smoothly altering the reward signal. It may be that in the experiments described in this thesis noise may be more sporadic in nature, when for example, clouds momentarily occlude sunlight from entering the lab.

[16] Personal Communication, to appear initially as UWE Learning Classifier System Group technical Report UWELCSG05-002

Secondly, Stone’s problems are single step while the ones described in this thesis are multi-step.

It would be interesting to attempt to characterise how robust X-TCS is with respect to noise. It would be a simple matter to shield the arena from varying background light, and then provide a ‘noise generator’ by varying the light levels. If X-TCS proves to be sensitive to noise this might suggest further investigation of TCS, perhaps adding weight to the need to investigate the automatic setting of parameters through self-adaptation.

6.2.4 Increasing Complexity

The problems to which X-TCS has been applied are very simple ones. In order for a robot to perform in complex problem domains, it will need to switch between many different competencies. In its current form, X-TCS has a ‘flat’ structure; all the classifiers are in the same population, and can all participate in every match set depending on their level on generality. The system learns not only the optimal behaviours, but also the optimal strategy for switching between them. For this reason it may be that X-TCS will not scale well as more goals need to be satisfied, demanding more complicated behaviours to be learned, perhaps with longer delay before external reward.

A number of approaches have been tried, as exemplified by the continual research interest in Robot Soccer, where, e.g., Bonarini et al. use their ‘BRIAN’ architecture to control the interaction of predefined behaviours through the use of a fuzzy control system [Bonarini2003]. Dorigo and Colombetti used a hierarchy of LCS to co-ordinate between learners that had been trained and then fixed [Dorigo1998]. Hierarchies of LCS – in this case XCS - were also used by Barry, and found suitable to solve the difficult Maze14 problem through the use of sub-goals to effectively increase the frequency of reward [Barry2001].

Again, Barry used a predefined hierarchical structure. Recent work by Gadanho [Gadano2002, Gadanho2003] uses predefined behaviours in a RL framework. A model of homeostatic variables based upon emotional states is presented to the neural networks that act as function approximators for a Q-Learner, which therefore has information about both external and internal environments. A value representing the 'well-being' of the system is derived from the differentially weighted sum of these internal variables, and this is fed to the learner as a reinforcement signal. They demonstrate the controller on a simulated Khepera robot. One problem with this approach is the sensitivity to the weighting of the homeostatic variables, which must be tuned by hand.

A most interesting direction for further research would be to investigate the ability to automatically generate hierarchies. Barry [ibid.] enumerates the benefits of a hierarchical approach as summarised below:

- Abstraction – solve simple problems first, and then build solutions to complex problems from these building blocks.
- Decomposition – divide complex problems into smaller problems that can be solved in isolation. Divide and Conquer.
- Reuse – once a behaviour has been learned, it need not be learned again.

By *automatically* generating hierarchies, X-TCS could become more widely applicable and generally adaptable. One possible approach for determining when to construct another layer in the hierarchy could be based upon the concept of Entropy as defined in Shannon's work on Information Theory [Shannon1948], and used in the work of Waldock et al. [Waldock2003] to construct a hierarchical fuzzy rule-base which the authors intend to apply to robotic learning, and as also used in the ID3 and C4.5 machine learning algorithms for decision tree construction [Quinlan1993].

6.2.5 Grounding – routes to semantic content

It will be recalled from the Introduction that ‘ungrounded’ intelligences operate purely syntactically. Semantic meaning can only be attributed from outside the system. Grounded systems, in which semantic content is intrinsic, are necessary for truly general applicability. In the multi-goal problems presented to the learners in this thesis the ‘energy character’ of the environmental input is linked to the internal state (i.e. energy level) of the learner through the action of the designer – semantic content has been assigned extrinsically. For X-TCS to be a more generally applicable solution to problems of robot control, this should not be necessary. Meaning should arise within the system itself, although of course at some level it must be necessary for ‘meaning’ to be externally assigned; a sensor must be supplied by the designer that measures some quantity, and this sensory reading must be made visible to the learner as a part of its environment.

There are a number of approaches which might be used to ground X-TCS. Hurst and Bull presented an extension of TCS in which the classifiers have a genotype encoding an artificial neural network and associated parameters for self-adaptation [Hurst2005]. Of these parameters, two encode the probability of adding or removing a node from the ANN encoded by the classifier. Classifiers can therefore grow or shrink in complexity to meet the demands of the problem, thereby having the potential to generalise over any subset of the environmental space. The ability to automatically adjust to problem complexity might further increase the generality of applicability of X-TCS. Some initial work using neural classifiers for multi-goal problems in simulation has already been undertaken by the author [Bull2002c].

An additional benefit may also be realised from a more complex classifier representation. In the current model used in this thesis, the condition of the classifier matches with some degree of generality the logical conjunction of the input variables. A hypothetical classifier might advocate the action ‘go ahead’ if the left sensor value is in the range [c1, c2] AND the forward sensor value is in the range [c3, c4] AND the right sensor value is in the range [c5, c6]. Unfortunately this representation allows for no comparison between the sensor values (e.g. go ahead if forward sensor value is bigger than both other sensors). The learner may be able to solve a problem at noon, but if faced with the same problem at dusk has to relearn the task! More complex condition semantics through the use of GP trees or artificial neural networks could solve this problem, once again increasing system utility.

An alternative approach to grounding the classifiers system might be to give it ‘memory’ in the fashion suggested by Wilson [Wilson1994]. Lanzi extended XCS in the same fashion that Cliff and Ross had used in ZCS [Cliff1994] to produce XCSM [Lanzi1998], in which a pre-specified number of characters could be used as internal memory. These were matched by an internal condition and might have their value changed by an internal action, both of which were extensions to the normal classifier representation. Lanzi used XCSM to show an increase in performance on simulated non-Markov mazes such as Woods101 where some states cannot be disambiguated from the environmental input, and in which therefore optimal performance is impossible without the ability to store information based upon previously encountered unambiguous states. This is similar to the use of ‘tags’ in Holland’s classifier systems [Holland1986] in which semantic networks could arise through rule-chaining and classifiers posted messages for matching on the internal message board along with the external environmental representation. In a multi-goal problem, an LCS might be able to evolve an optimum solution by using its ability to update an internal blackboard or change the value of memory characters, thereby recording information about past states in order to inform its decisions at a future time. Such an approach would be more grounded – the meanings of the symbols that predicate behaviour-switching are endogenous and hence the syntax-semantics barrier has been breached to some extent.

6.2.6 Accelerating Learning – Dyna and Eligibility Traces

One approach to increasing the speed of learning is to learn a model of the environment and make updates to the policy being followed on the basis of this model as well as on the basis of direct external RL updates, as in, e.g., Dyna-Q, presented in [Sutton1991, Sutton1998] which is an application of the general 'Dyna' architecture using Q-Learning. The indirect RL updates originating from the model may be viewed as a sort of simulated experience. Lanzi implemented the Dyna architecture with XCS as the learner and used it in some simple simulated mazes, reporting little improvement in such trivial problems [Lanzi1999]. In his implementation the model was simply composed of tuples storing information about state-action pairs, the subsequent state, and the reward received. Such a model is possible due to the discrete nature of the grid-world state-action space.

Lanzi notes that the size of the model would grow prohibitively as learning continued since a complete map of the environment would be produced, and therefore suggests the use of some form of generalization in the model. This would obviously be necessary to produce 'Dyna-X-TCS', as the state space is continuous. An additional instance of XCS could be used to learn the model with appropriate generalizations.

In the classifier systems used in this thesis, the process of reinforcement learning takes place through adapted versions of the implicit bucket brigade. Classifiers participating in an action set pass some reinforcement to their immediate predecessors, and will receive an update themselves from either their successors, or from an external reward. If we picture a chain of action sets leading from a start position to a goal state in a corridor, then we can see that after the first trial only the classifiers that caused the final action to be taken will be strengthened by the external reward. The second time the state two steps from the goal is reached some portion of this reward signal will flow back to the classifiers at that state. The flow of external reward, indicating true utility, is slow and step-wise. In a more complicated environment the flow will be slower still as many actions are possible at each state and many more state-action pairs must be assessed.

Given this delay in the flow of reinforcement in single-step temporal difference learning, the TD(λ) method is used in the RL community to improve the speed of prediction updates. A decaying log is kept of the states which can be updated and this has been shown to provide faster distribution of the reward to earlier states and faster convergence to optimal values. Drugowitsch and Barry have experimented with integrating eligibility traces into XCS, using a simple 7-step Finite State grid-world [Drugowitsch2005]. They find a reduction in performance in the resulting XCS(λ), which they suggest is due to overgeneral rules causing a propagation of errors resulting from incomplete exploration through the chain of rules to be updated. They show that a reduction in generality by lowering *P#* addresses this loss in performance. Since one of the major attractions of LCS over RL is the ability to generalize, this is unfortunate. However, they suggest that the problem may be less significant if the encoding allows for smooth changes in the degree of generalization. For this reason it might be expected that eligibility traces could add benefit and speed convergence in X-TCS or TCS, where the unordered-pair real number encoding does indeed allow such gradual adjustment. ANN classifier representations would be equally well-suited.

6.2.7 Whither TCS?

Given the poor performance of TCS on the dual objective task when compared to X-TCS, it might be tempting to ignore the former in favour of the latter in future work. However, as noted above, Stone has found that a strength-based LCS fared better than an accuracy-based LCS in very noisy environments, and that self-adaptation of parameters has been used to good effect which may help to solve the parameterisation problems. Finally Bull [Bull2005] suggests that the fundamental difference between accuracy-based and strength-based LCS may be in the level of fitness pressure such that there is a greater pressure in accuracy-based systems. It may be that further theoretical insight following from this work will enable simpler strength-based systems such as ZCS and its progeny TCS to perform as optimally and reliably as XCS and X-TCS.

References

[Apollonius] Apollonius of Rhodes, trans. Hunter, R. L., *Jason and the Golden Fleece: (The Argonautica)* (Oxford World's Classics) Oxford University Press; Reprint edition (August 1, 1998)

[Asada1996] Asada, M., Noda, S., Tawaratumida, S., and Hosoda, K. (1996) ‘Purposive behavior acquisition for a real robot by vision-based reinforcement learning.’ *Machine Learning*, 23:279-303.

[Barry2001] Barry, A.M. (2001), ‘A Hierarchical XCS for Long Path Environments’, in Spector, L. et al.(eds.), *Intl. Conference on Genetic and Evolutionary Computing (GECCO-2001).*

[Bonarini1994] Bonarini, A. (1994) ‘Evolutionary learning of general fuzzy rules with biased evaluation functions: competition and cooperation.’ *Proceedings of the IEEE WCCI - Evolutionary Computation.*

[Bonarini1996] Bonarini, A. (1996) ‘Evolutionary learning of fuzzy rules: competition and cooperation’, in: W. Pedrycz, Ed., *Fuzzy Modelling: Paradigms and Practice*, publ. Kluwer Academic Press, 265-283.

[Bonarini1997] Bonarini, A. , Basso F.(1997) ‘Learning to compose fuzzy behaviors for autonomous agents’. *International Journal on Approximate Reasoning*, vol. 17, no. 4

[Bonarini2003] Bonarini A., Invernizzi G., Labella T. H., Matteucci M. (2003), ‘An architecture to coordinate fuzzy behaviors to control an autonomous robot.’ *Fuzzy sets and systems.* 134(1), pp. 101-115.

[Booker1985] Booker, L. (1985) ‘Improving the Performance of Genetic Algorithms in Classifier Systems.’ In J.J. Grefenstette (ed.) *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, Lawrence Erlbaum Assoc., pp80-92

[Braitenberg1984] Braitenberg, V. (1984) *Vehicles: Experiments in Synthetic Psychology* Bradford Books; Reprint edition (February 7, 1986)

[Brooks1986] Brooks, R. A. (1986). ‘A robust layered control system for a mobile robot.’ *IEEE Journal of Robotics and Automation*, 2:14-23.

[Bull1998] Bull, L.(1998) ‘On ZCS in Multi-agent Environments.’ In A. E. Eiben, T. Baeck, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings Parallel Problem Solving From Nature (PPSN-V)*, volume 1498 of *Lecture Notes in Computer Science*, pages 471-480. Springer-Verlag, 1998.

[Bull1999] Bull, L. (1999) ‘On using ZCS in a Simulated Continuous Double-Auction Market.’ In W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela & R.E. Smith (eds) *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann, pp83-90.

[Bull2000a] Bull, L., Hurst, J. & Tomlinson, A. (2000) ‘Self-Adaptive Mutation in Classifier System Controllers.’ In J-A. Meyer, A. Berthoz, D.Floreano, H. Roitblatt & S.W. Wilson (eds) *From Animals to Animats 6 - The Sixth International Conference on the Simulation of Adaptive Behaviour,* MIT Press.

[Bull2000b] Bull, L. & Hurst, J. (2000) ‘Self-Adaptive Mutation in ZCS Controllers.’ In Cagnoni, S., Poli, R., Smith, G., Corne, D., Oates, M., Hart, E., Lanzi, P-L., Willem, E., Li, Y., Paecther, B. & Fogarty, T.C. (eds) *Real-World Applications of Evolutionary Computing: Proceedings of the EvoNet Workshops - EvoRob 2000*. Springer, pp339-346.

[Bull2002a] Bull, L. & Hurst, J. (2002) ‘ZCS Redux.’ *Evolutionary Computation* 10(2): 185-205.

[Bull2002b] Bull, L. (2002) ‘Lookahead and latent learning in ZCS’ In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 897-904, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[Bull2002c] Bull, L. & Studley, M. (2002) ‘Consideration of Multiple Objectives in Neural Learning Classifier Systems.’ In J.J. Merelo, P. Adamidis, H-G. Beyer, J-L. Fernandez-Villacanas & H-P. Schwefel (eds) *Parallel Problem Solving from Nature - PPSN VII.* Springer Verlag, pp549-557.

[Bull2005] Bull, L. (2005) ‘Two Simple Learning Classifier Systems’. In *Applications of Learning Classifier Systems Series: Studies in Fuzziness and Soft Computing*, Vol. 150 Eds. Bull, L & Kovacs, T. 2005, VIII

[Butz2001] Butz, M. & Wilson, S.W. (2001) ‘An Algorithmic Description of XCS’. *Lecture Notes in Computer Science*

[Butz2003a] Butz, M. V. & Goldberg, D. E. ‘Bounding the Population Size in XCS to ensure Reproductive Opportunities.’ IlliGAL report No. 2003009, February 2003.

[Butz2003b] Butz, M.V., Kovacs, T., Lanzi, P. L., & Wilson, S. W. (2003) ‘Toward a Theory of Generalization and Learning in XCS’ *IEEE Transaction on Evolutionary Computation*, Vol. 7, No. 6.

[Butz2003c] Butz, M. V., Sastry, K., Goldberg, D. E., (2003) ‘Strong, Stable, and Reliable Fitness Pressure in XCS due to Tournament Selection’, Illigal Technical Report 2003027

[Butz2004] Butz, M.V., Goldberg, D.E., & Lanzi, P.L. (in press) ‘Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems.’ *IEEE Transactions on Evolutionary Computation.*

[Capek1923] Capek, K., *R.U.R.* (first English publ. 1923) Dover Publications 2001

[Cliff1993] Cliff, D. T., Harvey, I., and Husbands, P. (1993). ‘Explorations in Evolutionary Robotics.’ *Adaptive Behaviour*, 2:73--110.

[Cliff1994] Cliff, D. and Ross, S. (1994) ‘Adding Temporary Memory to ZCS.’ *Adaptive Behaviour*, 3(2):101--150,

[Coello1999] Coello, C. A. (1999) ‘An Updated Survey of Evolutionary Multiobjective Optimization Techniques : State of the Art and Future Trends’. *1999 Congress on Evolutionary Computation*, pages 3--13

[Coello2000] Coello, C. A., (2001) ‘A Short Tutorial on Evolutionary Multiobjective Optimization’, In Zitzler, Deb, Thiele, Coello and Corne (editors), *First International Conference on Evolutionary Multi-Criterion Optimization*, Springer-Verlag, Lecture Notes in Computer Science No. 1993, pp. 21-40

[Connell1990] Connell, J. H. (1990)*Minimalist Mobile Robotics: A Colony Architecture for an Artificial Creature.* Academic Press

[Crabbe2001] Crabbe, F.L. (2001) Multiple Goal Q-Learning: Issues and Functions. *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation* (CIMCA)

[Deb2000] Deb, K., Pratap, A., Agrawal, S. and Meyarivan, T. (2000). A fast and elitist multiobjective genetic algorithm: NSGA-II. *Technical Report No. 2000001*. Kanpur: Indian Institute of Technology Kanpur, India.

[Deb2001] Deb, K. (2001) *Multi-Objective Optimization using Evolutionary Algorithms* public Wiley, ISBN: 0-471-87339-X, May

[Dorigo1998] Dorigo, M. and Colombetti, M. (1998) *Robot Shaping An Experiment in Behavior Engineering,* Intelligent Robotics and Autonomous Agents series, vol. 2. MIT Press,

[Drugowitsch2005] Drugowitsch, J., Barry, A.M. (2005), ‘XCS with Eligibility Traces’, University of Bath technical report ISSN 1740-9497

[Fikes1972] Fikes, R.E., Hart, P.E. , and Nilsson, N.J. (1972) ‘Learning and Executing Generalized Robot Plans.’ Artificial Intelligence, 3 251-288.

[Floreano1996] Floreano, D. and Mondada, F. (1996). ‘Evolution of homing navigation in a real mobile robot.’ *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26:396-407.

[Fogel1966] Fogel, L. J., Owens, A. J. and Walsh, M.J. (1966) *Artificial Intelligence Thorough Simulated Evolution*. John Wiley & Sons, Ltd, Chichester, U.K.

[Fogel1992] Fogel, D. B. (1992) *Evolving Artificial Intelligence*, Doctoral Dissertation, University of California, San Diego

[Fonseca1993] Fonseca C.M., Fleming P.J. (1993). ‘Genetic algorithms for multi-objective optimization: Formulation, discussion and generalization.’ In: S. Forrest (ed.), *Genetic algorithms: Proceedings of the Fifth International Conference*, Morgan Kaufmann, San Mateo, CA, 141-153.

[Fonseca1995] Fonseca, C. M. & Fleming, P. J. (1995) ‘An overview of evolutionary algorithms in multiobjective optimization.’, *Evolutionary Computation*, 3(1):1--16

[Gabor1998] Gabor, Z., Kalmar, Z., Szepesvari, C. (1998) ‘Multi-criteria reinforcement learning,’ , *Proceedings of the International Conference on Machine Learning*, Madison, WI

[Gadanho2002] Gadanho, S. C. and Custodio, L. (2002) ‘Learning behavior-selection in a multi-goal robot task.’ In NAISO ICAIS proceedings

[Gadanho2003] Gadanho, S. C. (2003) ‘Learning Behavior-Selection by Emotions and Cognition in a Multi-Goal Robot Task’ in *Journal of Machine Learning Research* , 4 (Jul): 385-412. The MIT Press

[Gambardella1995] Gambardella, L. M. & Dorigo, M. (1995) ‘Ant-Q: a reinforcement learning approach to the travelling salesman problem,’ In Prieditis, A. and Russell, S. (Eds.), *Proceedings of ML-95, Twelfth International Conference on Machine Learning*, pp. 252—260, Morgan Kaufmann.

[Goldberg1989] Goldberg D, (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley

[Gordon1999] G. J. Gordon. (1999) ‘Approximate Solutions to Markov Decision Processes.’ PhD thesis, Carnegie Mellon University.

[Harnad1990] Harnad, S. (1990) ‘The Symbol Grounding Problem.’ *Physica* D 42:pp. 335-346.

[Holland1975] Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

[Holland1976] Holland, J.H. (1976) ‘Adaptation.’ In R. Rosen and F. M. Snell, editors, *Progress in Theoretical Biology*. New York: Plenum, 1976.

[Holland1978] Holland, J.H. and Reitman, J. S. (1978) ‘Cognitive Systems Based on Adaptive Algorithms’, in D.A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic Press, NY.

[Holland1986] Holland, J. H. (1986) ‘Escaping Brittleness: The possibilities of General-purpose Learning Algorithms Applied to Parallel Rule-Based Systems’, in Mitchell, T.M., Michalski, R. S., and Carbonell, J.G. (eds.), *Machine Learning, An Artificial Intelligence Approach*, Vol. II, ch. 20, 593-623, Morgan Kaufmann.

[Holland1997] Holland, O. E. (1997) ‘Grey Walter: The Pioneer of Real Artificial Life’, *Proceedings of the 5th International Workshop on Artificial Life*, C. Langton Editor, MIT Press, Cambridge, p34-44.

[Hornby1999] Hornby, G. S., Fujita, M., Takamura, S., Yamamoto, T., & Hanagata, O. (1999). ‘Autonomous evolution of gaits with the Sony quadruped robot.’ In *Proceedings of the Genetic and Evolutionary Computation Conference*. Morgan Kaufmann.

[Hornby2000] Hornby, G. S., Fujita, M., Takamura, S., Yamamoto, T., Hanagata, O. and Fujita, M. (2000) ‘Evolving robust gaits with Aibo.’ In *Proceedings of ICRA-2000*.

[Hubel1988] Hubel, D. H. (1988) *Eye, brain, and vision*. Scientific American Library

[Hurst2001] Hurst, J. & Bull, L. (2001) ‘A Self-Adaptive Classifier System.’ In P-L. Lanzi, W. Stolzmann & S.W. Wilson (eds) *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop*, Springer, pp70-79.

[Hurst2002a] Hurst, J., Bull, L. & Melhuish, C. (2002) ‘TCS Learning Classifier System Controller on a Real Robot.’ In J.J. Merelo, P. Adamidis, H-G. Beyer, J-L. Fernandez-Villacanas & H-P. Schwefel (eds) *Parallel Problem Solving from Nature - PPSN VII.* Springer Verlag, pp588-600.

[Hurst2002b] Hurst, J., Bull, L. & Melhuish, C. (2002) ‘ZCS and TCS Learning Classifier System Controllers on Real Robots.’ UWELCSG02-002. (University of the West of England Technical report, available for download from http://www.cems.uwe.ac.uk/lcsg/)

[Hurst2003] Hurst, J. (2003) *Learning Classifier Systems in Robotic Environments* PhD Thesis, Faculty of Computing, Engineering, and the Mathematical Sciences, University of the West of England

[Hurst2005] Hurst, J. & Bull, L. (2005) ‘A Neural Learning Classifier System with Self-Adaptive Constructivism for Mobile Robot Control.’ *Artificial Life* (in press).

[Jakobi1995] Jakobi, N, Husbands, P. and Harvey, I. (1995) ‘Noise and the reality gap: The use of simulation in evolutionary robotics.’ In F. Moran, A. Moreno, J. Merelo, and P. Chacon, editors, Advances in Artificial Life: Proc. 3rd European Conf. on Artificial Life, 704--720. Lecture Notes in Artificial Intelligence 929.

[Japkowicz2002] Japkowicz, N. and Stephen, S. ‘The Class Imbalance Problem : A Systematic Study’ *Intelligent Data Analysis*, 6(5): 429-450, November 2002.

[Kaelbling1996] Kaelbling, L.P., Littman, L.M. and Moore, A.W. (1996) ‘Reinforcement learning: a survey,’ *Journal of Artificial Intelligence Research*, vol. 4, pp. 237--285.

[Katagami2000] D. Katagami and S. Yamada: ‘Interactive Classifier System for Real Robot Learning’, *IEEE International Workshop on Robot-Human Interaction (ROMAN-2000)*, pp.258-263, Osaka, Japan (September, 2000)

[Katagami2001] Katagami, D. and Yamada, S. (2001) ‘Real Robot Learning with Human Teaching’, *The 4th Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems*, pp.263-270

[Katagami2003] Katagami, D. and Yamada, S. (2003) ‘Teacher’s Load and Timing of Teaching based on Interactive Evolutionary Robotics’, *IEEE International Conference in Robotics and Automation (CIRA2003)*, pp.1096-1101

[Kharbat2005] Kharbat, F, Bull, L. & Odeh, M. (2005) ‘Tournament and Roulette Wheel Selection in XCS’, UWELCSG05-003. (University of the West of England Technical report, available for download from http://www.cems.uwe.ac.uk/lcsg/)

[Kleene1967] Kleene, S.C. 1967. *Mathematical Logic*. New York: Wiley.

[Kovacs1999] Kovacs, T. (1999) ‘Deletion Schemes for Classifier Systems.’ *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, ed. Banzhaf et al. Morgan Kaufmann, July.

[Koza1992] Koza J.R. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

[Langdon2002] Langdon, W. and Poli, R. (2002) ‘Tutorial on Foundations of Genetic Programming.’ In *GECCO 2002 Tutorials.*

[Lanzi1997] Lanzi P.L. (1997) ‘A Model of the Environment to Avoid Local Learning (An Analysis of the Generalization Mechanism of XCS).’ *Technical Report 97.46*, Politecnico di Milano. Department of Electronic Engineering and Information Sciences.

[Lanzi1998] Lanzi, P. L. (1998) ‘Adding Memory to XCS.’ in *Proceedings of the IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on Evolutionary Computatio*n, May 4-9 Anchorage (AL), pages 609-614. IEEE Press.

[Lanzi1999] Lanzi, P. L. (1999). ‘An analysis of generalization in the XCS classifier system.’ *Evolutionary Computation*, 7 (2), 125--149.

[Llora2003] Llora, X., Goldberg, D. E. (2003) ‘Bounding the effect of noise in Multiobjective Learning Classifier Systems.’ *Evolutionary Computation* 11(3): 278-297

[Mahadevan1991] S. Mahadevan and J. H. Connell, (1991) ‘Automatic Programming of Behaviour-based Robots Using Reinforcement Learning’, *Artificial Intelligence,* Vol. 55, pages 311-365.

[Mannor2004] Mannor, S. and Shimkin, N. (2004), A Geometric Approach to Multi-Criterion Reinforcement Learning. Journal of Machine Learning Research 5: 325-360

[Mariano1999] Mariano C.E., Morales E. (1999), ‘MOAQ and ant-Q algorithm for multiple objective optimization problems’, *Proceeding of the Genetic and Evolutionary Computation Conference,* Orlando, Florida, USA, 13-17 July, vol. 1, 894-901.

[Marocco2002] Marocco, D., and Floreano, D. (2002), ‘Active Vision and Feature Selection in Evolutionary Behavioral Systems’, In B. Hallam, D. Floreano, J. Hallam, G. Hayes and J.-A. Meyer (Eds.), *From animals to animats 7 - The Seventh International Conference on the Simulation of Adaptive Behavior*, The MIT Press

[Matellan1998] Matellan, V., Fernandez, C., & Molina, J. (1998). ‘Genetic learning of fuzzy reactive controllers.’ *Robotics and Autonomous Systems*, 25, 33-41

[Minsky1961] Minsky, M. (1961) ‘Steps toward artificial intelligence.’ Proc. IRE 49

[Newell1957] Newell, A., Shaw, J.C., and Simon, H. A. (1957) ‘Empirical Explorations of the Logic Theory Machine’, *Proceedings of the Western Joint Computer Conference*, pp. 218-239.

[Newell1976] Newell, A. and Simon, H. A. (1976) ‘Computer science as empirical enquiry: symbols and search’, *Communications of the ACM*, 19(3), 113--126

[Nilsson1984] Nilsson, N., (1984) (ed.), ‘Shakey the Robot’, Technical Note 323, SRI International, Menlo Park, CA

[Nolfi1994] S. Nolfi, D. Floreano, O. Miglino, and F. Mondada. (1994) ‘How to evolve autonomous robots: Different approaches in evolutionary robotics’. In R. Brooks and P. Maes, editors, *Artificial Life IV*, pages 190--197. MIT Press/Bradford Books

[Nolfi2000] Nolfi, S. and Floreano, D. (2000) *Evolutionary Robotics The Biology, Intelligence, and Technology of Self-Organizing Machines* MIT Press/Bradford Books

[Orriols2005a] Orriols, A. and Bernadó-Mansilla, E. 'The Class Imbalance Problem in UCS Classifier System: Fitness Adaptation.' In Proceedings of the 2005 Congress on Evolutionary Computation, 2005, IEEE, (in press).

[Orriols2005b] Orriols, A. and Bernadó-Mansilla, E. 'The Class Imbalance Problem in Learning Classifier Systems: A Preliminary Study.' In Proceedings of the 8th International Workshop on Learning Classifier Systems, 2005, Springer, (in press)

[Parr1998] Parr, R. E. (1998) *Hierarchical Control and Learning for Markov Decision Processes.* PhD thesis, University of California, Berkeley, CA

[Quinlan1993] Quinlan, J. R.(1993) *C4.5: Programs for Machine Learning*, publ. Morgan Kauffman

[Rechenberg1965] Rechenberg. (1965) *Cybernetic Solution Path of an Experimental Problem*. Ministry of Aviation, Royal Aircraft Establishment, U.K.

[Rechenberg1973] Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution.* Stuttgart: Frommann-Holzboog.

[Rodriguez-Vazquez1993] Rodriguez-Vazquez, K., Fonseca, C., & Fleming, P. (1997) ‘Multiobjective Genetic Programming : A Nonlinear System Identification Application, *Late Breaking Papers at the Genetic Programming 1997 Conference*, pages 207--212, Stanford University, California, July

[Rosenblatt1958] Rosenblatt, F. (1958), ‘The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain’, Cornell Aeronautical Laboratory, *Psychological Review*, v65, No. 6, pp. 386-408.

[Rummery1995] Rummery, G. A. (1995) *Problem Solving With Reinforcement Learning.* PhD thesis, University of Cambridge

[Santamaria1998] Santamaria, J. C., Sutton, R. C., and Ram, A. (1998) ‘Experiments with reinforcement learning in problems with continuous state and action spaces.’ in *Adaptive Behaviour,* 6(2)

[Schaffer1985] Schaffer, J.D. (1985) ‘Multi-objective optimization with vector evaluated genetic algorithms.’ In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, J.J. Grefenstette, Ed., Pittsburg, PA, July 24-26, pp. 93-100

[Searle1980] Searle, J. (1980). ‘Minds, Brains, and Programs.’ *Behavioral and Brain Sciences* 3, 417-424.

[Shannon1948] Shannon C. E. (1948) ‘A mathematical theory of communication’, Bell System Technical Journal, vol. 27, pp.379-423 and 623-656, July and October

[Shelley1818] Shelley, M. W. (1818) , *Frankenstein: Or the Modern Prometheus* (Penguin Classics) publ. Penguin Books; Reissue edition (April 1, 2003)

[Shortliffe1973] Shortliffe, E. H., Axline, S.G., Buchanan, B.G., Merigan, T.C., & Cohen, S. N. (1973) ‘An artificial intelligence program to advise physicians regarding antimicrobial therapy.’ *Computers and Biomedical Research* 6:544-560

[Skinner1953] Skinner, B. F. (1953) *Science and Human Behavior*. New York: Macmillan.

[Smart2000] W.D. Smart and L.P. Kaelbling (2000.) ‘Practical Reinforcement Learning in Continuous Spaces’ In *Proceedings of the Seventeenth International Conference on Machine Learning,* pp. 903-910

[Smart2002a] W. D. Smart and L. P. Kaelbling, (2002) ‘Effective Reinforcement Learning for Mobile Robots,’ *International Conference on Robotics and Automation*, May 11-15

[Smart2002b] W. D. Smart. (2002) *Making Reinforcement Learning Work on Real Robots* PhD Thesis, Department of Computer Science, Brown University

[Smith1980] Smith, S. F. (1980) *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, Computer Science Department, University of Pittsburgh.

[Srinivas1994] Srinivas, N. and Deb, K. (1994) ‘Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms’, in *Evolutionary Computation*, vol. 2, no 3, pps. 221-248

[Stephens1999] Stephens, C. and Waelbroeck, H. (1999) ‘Schemata Evolution and Building Blocks.’ In *Evolutionary Computation* 7(2)

[Stolzmann1998] Stolzmann, W. (1998) ‘Anticipatory Classifier Systems’, in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, editors, Koza, Banzhaf et al. , pps. 658—664,   Publisher Morgan Kaufmann

[Stolzmann1999] Stolzmann, W. (1999) ‘Latent Learning in Khepera Robots with Anticipatory Classifier Systems.’ In A.S. Wu (Ed.), *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Programme*, pp. 290-297.

[Stone2003] Stone, C. & Bull, L. (2003) ‘For Real! XCS with Continuous-Valued Inputs.’ *Evolutionary Computation* 11(3) 299-336.

[Sutton1991] Sutton, R. (1991) ‘Dyna, an integrated architecture for learning, planning, and reacting.’ *SIGART Bulletin*, 2:160–163

[Sutton1998] Sutton R.A., Barto A.G. (1998) *Reinforcement Learning: An Introduction* Publ. MIT Press, Cambridge, MA

[Sutton1999] Sutton, R., Precup, D., and Singh, S. (1999) ‘Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning.’ In *Artificial Intelligence Journal*, vol. 112, pp. 181-211.

[Syswerda1989] Syswerda, G. (1989) ‘Uniform Crossover in Genetic Algorithms’, in *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2-9, Morgan Kaufmann Publishers.

[Thieberger1955] Thieberger, F. (1955). *The great Rabbi Loew of Prague: His life and work and the legend of the golem.* London: Horovitz Publishing Co.

[Thrun1993] Thrun, S. and Schwartz, A. (1993) ‘Issues in Using Function Approximation for Reinforcement Learning’ In M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, editors, *Proceedings of the Connectionist Models Summer School*, pp. 255-263, Hillsdale, NJ

[Tomlinson1998] Tomlinson A. and Bull, L. (1998) ‘A Corporate Classifier System.’ In A. E. Eiben, T. Bäck, M. Shoenauer, and H.-P. Schwefel, editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving From Nature -- PPSN V,* number 1498 in LNCS, pages 550-559. Springer Verlag

[Uchibe1997] Uchibe, E., Asada, M., Hosoda, K. (1997) ‘Vision Based State Space Construction for Learning Mobile Robots in Multi Agent Environments.’ *Proc. of Sixth European Workshop on Learning Robots* (EWLR-6) 33-41

[Valenzuela-Rendon1991] Valenzuela-Rendon, M. (1991) ‘The Fuzzy Classifier System: a Classifier System for Continuously Varying Variables.’ In L. Booker & R. Belew (eds) *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, pp346-353.

[Waldock2003] Waldock, A., Carse, B. and Melhuish, C. (2003) ‘A Hierarchical Fuzzy Rule-based Learning System based on an Information Theoretic Approach’ European Society For Fuzzy Logic and Technology International Conference in Fuzzy Logic and Technology pp534-539, September 10 - 12, 2003 Zittau, Germany.

[Walker2003] Walker J, Garrett S and Wilson W (2003). ‘Evolution for Real Robots: A Structured Survey of the Literature’. *Adaptive Behavior* Vol 11(3): 179-203.

[Watkins1989] Watkins, C. (1989*). Learning from Delayed Rewards*, Thesis, University of Cambridge, England.

[Wilson1987] Wilson, S. W. (1987)’Classifier Systems and the Animat Problem’, *Machine Learning, 2,* 199-228

[Wilson1994] Wilson, S. W. (1994) ‘ZCS: A zeroth level classifier system.’ In *Evolutionary Computation*, 2(1):1-18

[Wilson1995] Wilson, S. (1995) ‘Classifier Fitness based on Accuracy.’ *Evolutionary Computation*, 3(2):149--175.

[Wilson1998] Wilson, S. W. (1998). ‘Generalization in the XCS classifier system.’ In *Proceedings of the Third Annual Genetic Programming Conference*, J. Koza et al.(eds.), San Francisco, CA: Morgan Kaufmann, 665-6

[Wilson2000] Wilson, S.W. (2000) ‘Get real! XCS with continuous-valued inputs’ in Lanzi, P. L., Stolzmann, W., and Wilson, S. W., eds. *Learning Classifier Systems. From Foundations to Applications Lecture Notes in Artificial Intelligence* (LNAI-1813) Berlin: Springer-Verlag

[Wilson2001] Wilson, S. W. (2001) ‘ Mining oblique data with XCS’ In Lanzi, P. L., Stolzmann, W., and S. W. Wilson (Eds.), *Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000),* Lecture Notes in Artificial Intelligence (LNAI-1996). Berlin: Springer-Verlag

[Winfield2000] Winfield AFT and Holland OE, (2000) ‘The application of wireless local area network technology to the control of mobile robots’, *Journal of Microprocessors and Microsystems* (Elsevier), Vol 23/10, pp 597-607

[Winfield2003] Winfield AFT, (2003) ‘Linux: an Embedded Operating System for Mobile Robots’, invited paper in *IEE Embedded and Real-time Systems Professional Network colloquium on Developing Embedded Real-Time Systems*, London, May 2003.

[Zitzler1999] Zitzler. E., Deb, K., & Thiele, L. (1999) ‘Comparison of multiobjective evolutionary algorithms: Empirical results,’ *Tech. Rep. 70,* Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland

Glossary[17]

Agent A system that is embedded in an environment, and takes actions to change the state of the environment. Examples include mobile robots, software agents, or industrial controllers.

Average-Reward Methods A framework where the agent's goal is to maximize the expected payoff per step. Average-reward methods are appropriate in problems where the goal is maximize the long-term performance. They are usually much more difficult to analyze than discounted algorithms.

Building Block A pattern of genes in a contiguous section of a chromosome which, if present, confers a high fitness to the individual. According to the building block hypothesis, a complete solution can be constructed by crossover joining together in a single individual many building blocks which where originally spread throughout the population.

Chromosome Normally, in genetic algorithms the bit string which represents the individual. In genetic programming the individual and its representation are usually the same, both being the program parse tree. In nature many species store their genetic information on more than one chromosome.

Coevolution Two or more populations are evolved at the same time. Often the separate populations compete against each other.

Convergence Tendency of members of the population to be the same. May be used to mean either their representation or behaviour are identical. Loosely a genetic algorithm solution has been reached.

Crossover Creating a new individual's representation from parts of its parents' representations.

[17] The definitions in this glossary have been drawn from the following sources:

http://en.wikipedia.org/

http://www-all.cs.umass.edu/rlr/terms.html

http://www.cs.bham.ac.uk/~wbl/thesis.glossary.html

Discount Factor A scalar value between 0 and 1 which determines the present value of future rewards. If the discount factor is 0, the agent is concerned with maximizing immediate rewards. As the discount factor approaches 1, the agent takes more future rewards into account. Algorithms which discount future rewards include Q-learning and TD(lambda).

Dynamic Programming (DP) is a class of solution methods for solving sequential decision problems with a compositional cost structure.

Elitist An elitist genetic algorithm is one that always retains in the population the best individual found so far. Tournament selection is naturally elitist.

Environment The external system in which an agent is 'embedded', and which it can perceive and act upon.

Epistasis A term from biology used to denote that the fitness of an individual depends upon the interaction of a number of their genes. In genetic algorithms this would be indicated by the fitness containing a non-linear combination of components of the string.

Evolution Strategy Each point in the search space is represented by a vector of real values. In the original Evolution Strategy, (1+1)-ES, the next point to search is given by adding gaussian random noise to the current search point. The new point is evaluated and if better the search continues from it. If not the search continues from the original point. The level of noise is automatically adjusted as the search proceeds.

Evolutionary Programming like Evolution Strategy produces new children by mutating at random from a single parent solution. The analogue components (e.g. the connection weights when applied to artificial neural networks) are changed by a gaussian function whose standard deviation is given by a function of the parent's error called its temperature. Digital components (e.g. presence of a hidden node) are created and destroyed at random.

Fitness Function A process which evaluates a member of a population and gives it a score or fitness. In most cases the goal is to find an individual with the maximum (or minimum) fitness.

Genetic Algorithm (GA) A population containing a number of trial solutions each of which is evaluated (to yield a fitness) and a new generation is created from the better of them. The process is continued through a number of generations with the aim that the population should evolve to contain an acceptable solution. GAs are characterised by representing the solution as an (often fixed length) string of digital symbols, selecting parents from the current population in proportion to their fitness (or some approximation of this) and the use of crossover as the dominate means of creating new members of the population. The initial population may be created at random or from some known starting point.

Genetic Operator An operator in a genetic algorithm or genetic programming, which acts upon the chromosome to produce a new individual. Example operators are mutation and crossover.

Genetic Programming A subset of genetic algorithms. The members of the populations are the parse trees of computer programs whose fitness is evaluated by running them. The reproduction operators (e.g. crossover) are refined to ensure that the child is syntactically correct (some protection may be given against semantic errors too).

Learning Classifier System An extension of genetic algorithms in which the population consists of a co-operating set of rules (i.e. a rulebase) which are to learn to solve a problem given a number of test cases. Between each generation the population as a whole is evaluated and a fitness is assigned to each rule using the bucket-brigade algorithm or other credit sharing scheme (e.g. the Pitt scheme). These schemes aims to reward or punish rules which contribute to a test case according to how good the total solution is by adjusting the individual rules fitness.

Markov Decision Process (MDP) A probabilistic model of a sequential decision problem, where states can be perceived exactly, and the current state and action selected determine a probability distribution on future states. Essentially, the outcome of applying an action to a state depends only on the current action and state (and not on preceding actions or states).

Model The agent's view of the environment, which maps state-action pairs to probability distributions over states. Note that not every reinforcement learning agent uses a model of its environment.

Model-Free Algorithms These directly learn a value function without requiring knowledge of the consequences of doing actions. Q-learning is the best known example of a model-free algorithm.

Monte Carlo Methods A class of methods for learning of value functions, which estimates the value of a state by running many trials starting at that state, then averages the total rewards received on those trials.

Mutation Arbitrary change to representation, often at random. GP subtrees or nodes of trees are replaced at random, real numbers are increased or decreased, letters in an alphabet are replaced with other members of the set of letters.

Panmictic Descriptive of an evolutionary system in which no constraints are placed on mating. Such restrictions might be due to analogues of spatial distribution or mate choice.

Policy The decision-making function (control strategy) of the agent, which represents a mapping from situations to actions.

Reproduction Production of new member of population from existing members. May be used to mean an exact copy of the original member.

Reinforcement Learning (RL) is learning from interaction with an environment, from the consequences of action, rather than from explicit teaching. RL become popular in the 1990s within machine learning and artificial intelligence, but also within operations research and with offshoots in psychology and neuroscience.

Reward A scalar value which represents the degree to which a state or action is desirable. Reward functions can be used to specify a wide range of planning goals (e.g. by penalizing every non-goal state, an agent can be guided towards learning the fastest route to the final state).

Roulette Wheel Selection The simplest selection scheme is roulette-wheel selection, also called stochastic sampling with replacement. This is a stochastic algorithm in which individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness. A random number is generated and the individual whose segment spans the random number is selected. This technique is analogous to a roulette wheel with each slice proportional in size to the fitness of the individuals.

Sensor Agents perceive the state of their environment using sensors, which can refer to physical transducers, such as ultrasound, or simulated feature-detectors.

Simulated Annealing Search technique where a single trial solution is modified at random. An energy is defined which represents how good the solution is. The goal is to find the best solution by minimising the energy. Changes which lead to a lower energy are always accepted; an increase is probabilistically accepted. The probability is given by exp(-Delta E/kT). Where Delta E is the change in energy, k is a constant and T is the Temperature. Initially the temperature is high corresponding to a liquid or molten state where large changes are possible and it is progressively reduced using a cooling schedule so allowing smaller changes until the system solidifies at a low energy solution.

State This can be viewed as a summary of the past history of the system, that determines its future evolution.

Stochastic Random or probabilistic but with some direction. For example the arrival of people at a post office might be random but average properties (such as the queue length) can be predicted.

Supervised Learning is a machine learning technique for creating a function from training data. The training data consist of pairs of input objects (typically vectors), and desired outputs. The output of the function can be a continuous value (called regression), or can predict a class label of the input object (called classification). The task of the supervised learner is to predict the value of the function for any valid input object after having seen a number of training examples (i.e. pairs of input and target output). To achieve this, the learner has to generalize from the presented data to unseen situations in a ‘reasonable’ way

TD (Temporal Difference) Algorithms A class of learning methods, based on the idea of comparing temporally successive predictions. Possibly the single most fundamental idea in all of reinforcement learning.

Tournament Selection A mechanism for choosing individuals from a population. A group (typically between 2 and 7 individuals) are selected at random from the population and the best (normally only one, but possibly more) is chosen.

Unsupervised Learning The area of machine learning in which an agent learns from interaction with its environment, rather than from a knowledgeable teacher that specifies the action the agent should take in any given state.

Value Function A mapping from states to real numbers, where the value of a state represents the long-term reward achieved starting from that state, and executing a particular policy. The key distinguishing feature of RL methods is that they learn policies indirectly, by instead learning value functions. RL methods can be constrasted with direct optimization methods, such as genetic algorithms (GA), which attempt to search the policy space directly.