1

Introduction

The appeal of Machine Learning (ML) lies in the idea of computers teaching themselves to solve problems, rather than relying on humans to specify their every move. Relying on humans to hard-wire behaviour is limiting because of the obvious difficulties of anticipating any number of situations in advance, particularly in a changing world. But further, we often simply do not know *how* it is we do what we do, and so cannot specify it to a computer. And, of course, there are any number of problems we have trouble solving ourselves in the first place. So there are limits to the complexity of the problems we can address by building hard-wired solutions.

We can sidestep these problems by, instead of building hard-wired machines, building learning machines which generate desired behaviour on their own. That is, by solving one problem (that of building a learning machine), we obtain a solution to many problems (those to which we apply our machine). This leaves us the task of specifying what the desired behaviour of the learning machine should be, but the assumption is that this should be easier than programming this behaviour. Of course, the problem of building learning machines which can handle complex tasks is itself a most difficult undertaking; it is generally easier to solve the problem at hand than to build a learning machine which is capable of solving it.

Nonetheless, ML promises rewards commensurate with its difficulties. It has already been suggested that learning machines promise to repay the effort invested in them, and to do things which humans cannot. But research into machine learning is not only a means of addressing engineering problems. It is also ultimately a part of the endeavour to understand intelligence and create machine intelligence, and so bears on philosophy, psychology, cognitive science, ethology and other fields. To sum up, ML involves very difficult problems, but holds tremendous promise, both in terms of extending the utility of computers and in the study of intelligence.

This book is concerned with Learning Classifier Systems (LCS), a form of machine learning system introduced in the 1970s by John Holland, which typically incorporates both Evolutionary Algorithms and Reinforcement Learning

algorithms. Like ML in general, building successful LCS involves terribly difficult problems, but successful LCS hold tremendous promise. Put another way, if the goals of classifier systems research were met, many of the goals of machine learning would be met.

1.1 Two Example Machine Learning Tasks

This section introduces two example machine learning tasks and suggests a few ways in which we might approach them. In §1.5 we introduce learning classifier systems which can be applied to these tasks.

Example: Classifying Mushrooms

Consider the problem of enabling a computer to distinguish between poisonous and edible mushrooms. Human experts learn through experience how to distinguish the two, and it is difficult to find simple rules with good predictive power. Thus, it seems reasonable to apply machine learning to the problem of finding good, simple rules which distinguish the two cases. The standard approach is to train a machine learning system on examples of preclassified mushrooms, that is, a set of example mushrooms which have already been classified by a human expert. For example, we might use a database of descriptions of 100 poisonous and 100 edible mushrooms. Rather than present the computer with a raw image of a mushroom, we can extract certain information, e.g., concerning colour, size, shape and so on. The ML system can then teach itself how to classify the mushrooms, using the human expert's classifications for reference. Once the system has achieved good performance on this training set, we can test it on a set of previously unseen mushrooms (again, preclassified by a human expert) to see how well it generalises from the examples it has seen to new cases. If it performs well on this test, we may be willing to trust its judgement concerning mushrooms which have not been classified by a human expert. In other words, the ML system learns to duplicate a human expert's skill, and to generalise from its limited experience. One use for such a system is to evaluate mushrooms without the need to consult human experts, who may be few and far between. Another use is for the human expert to consider the decision-making rules found by the system, which may differ from those used by humans and which may offer some insight into the problem of mushroom classification.

A great variety of ML systems could conceivably be applied to this problem, but let us consider a system which, in teaching itself how to classify mushrooms, explicitly entertains many competing hypotheses and tests them against the available data. If the data supports a hypothesis, this suggests it should be retained (and perhaps used as the basis for new hypotheses). If the data does not support a hypothesis, this suggests it should be discarded. This approach seems fairly obvious and fundamental. (We might want to bias the

system to be more likely to misclassify non-poisonous mushrooms as poisonous than the other way around, but we won't go into such details here.)

One issue in developing such a system is how to represent hypotheses. An obvious and fundamental approach is to use *condition-action rules*, such as these:

IF	small **AND** green	**THEN**	edible
IF	(small **AND** green) **OR** has-spots	**THEN**	poisonous
IF	small **AND** pink	**THEN**	poisonous
IF	small **AND** pink	**THEN**	edible

Example: Simulating a Frog

As another example, consider this problem: given sensors and effectors, provide a self-adapting control system for a simulated frog which learns only by trial and error, that is, from rewards and punishments from its simulated environment. The frog's objectives are to maximise the number of flies it eats, minimise its energy use, and to avoid being eaten itself.

This problem differs significantly from the mushroom classification task in that the system learns only from rewards and punishments, not from preclassified examples. Another difference is that in this case the learner *interacts* with a problem environment; the frog's actions influence the environment. As with the mushroom classification task, it may be difficult for humans to produce a system which effectively maximises the rewards and minimises the punishments the frog receives. To address the problem, we might again reasonably attempt to develop a machine learning system which entertains and evaluates multiple competing hypotheses, and which represents them with condition-action rules, for example:

IF	small **AND** dark **AND** buzzes	**THEN**	eat-it
IF	large **AND** white **AND** has-beak	**THEN**	hide

Classifier Systems

The broad approach suggested for both problems in this section is that taken by classifier systems, so-called because they learn condition-action rules called *classifiers*.

1.2 Types of Task

1.2.1 Supervised and Reinforcement Learning

The two examples we've just seen are meant to illustrate two kinds of task to which we can apply LCS. The mushroom classification task is meant to

illustrate a *pattern classification* or *data mining* task, while the frog controller is an on-line *control* task.

Each task is suitable for a different learning paradigm, that is, a different model of interaction between the learning agent and the task. In the mushroom classification task we have a training set of pre-classified exemplars, making Supervised Learning (SL) feasible. For the frog controller task, however, we have only certain conditions which are good (eating flies) or bad (being eaten), and so the task must be modelled as a Reinforcement Learning (RL) task. Learning paradigms are covered in Appendix C. Classifier systems are suitable for both paradigms, but this work focuses on RL.

1.2.2 Sequential and Non-sequential Decision Tasks

The two tasks illustrate a second important distinction to be made in the space of tasks. The mushroom classification task is non-sequential, that is, the action taken by the learner on one time step does not influence what inputs it receives in the future. (The classification it gives at time t has no influence on which mushroom it sees at time $t + 1$.) In contrast, the frog simulation task is sequential; an action taken at time t may well influence what inputs are received in the future. For example, at time t the frog may choose between hopping into a pond and hopping into some tall grass, each of which will lead it through a different sequence of states in the future. Sequential tasks are in general more difficult because they require the learner to consider the long-term consequences of its actions. This work studies the application of classifier systems to both types of task.

1.3 Two Challenges for Classifier Systems

Our suggestion that the system entertain competing hypotheses in the form of condition-action rules is a rather vague specification. Among the major questions it leaves open are:

How do we choose between rules? The reader may have noticed that some of the rules in the mushroom classification example contradict each other. When this happens some kind of *conflict resolution* must be performed. Good conflict resolution relies on our ability to discriminate between good and bad rules, that is, to assess the utility of rules. Ideally, the means by which we do so should be independent of the domain in which the system operates; it is desirable to use the same methods for evaluating rules and resolving conflicts between them in all domains, rather than having to tailor them to each domain.

Where do rules come from? A classifier system needs some automated means of generating rules, and employs a *generate-and-test* approach. To optimise this process, rather than generate random rules, we would like to

base new rules on existing high-quality rules. Thus we have two processes which must choose between rules; conflict resolution, and the selection of rules on which to base new rules. The generation of rules must be independent of human intervention, and, ideally, independent of the domain in which the system operates; to minimise human input it is desirable to use the same rule-discovery techniques in searching for good rules in all domains, rather than have to tailor them to each problem.

These two questions are major issues for LCS research. They have been the subject of much research over the years, and will be for many years to come. The answers to these questions can define vastly different systems, with vastly different characteristics.

Because this work is concerned with RL classifier systems, and because it will borrow much from existing work on RL, the two challenges above are recast below in terms of reinforcement learning. From this point on, we will assume tasks and learning agents which fall under the RL heading, unless otherwise noted.

1.3.1 Problem 1: Learning a Policy from Reinforcement

The challenge of choosing between conflicting rules is really the problem of learning a *policy* – a mapping from states to actions – which tells the learning agent how to behave. Consequently, this is sometimes known as the *control* problem. The goal of a learning agent is to find an *optimal policy* (§F.2), that is, one which, over the long run, maximises the rewards and minimises the punishments it receives.

One aspect of searching for good policies is the need to evaluate policies, or parts of policies, e.g., to evaluate how good it is to take an action a in a state s. This is also known as the *prediction* problem, since it can be seen as the task of predicting the outcome of behaviour. Methods of addressing the prediction problem are covered in Appendix E, and the interaction of control and prediction is discussed in §F.3 and §G.7.1.

1.3.2 Problem 2: Generalisation

To scale up to large tasks learning methods must exploit regularities in the task, in their representation of it and their operation. Consequently, it is not enough to search for a good policy; to be effective, a learning system must exploit regularities in doing so. One way to exploit regularities is to generalise over (alias) aspects of the task. Appendix D discusses this subject in more depth.

1.4 Solution Methods

This work is concerned with two basic approaches to learning policies from reinforcement, and are introduced in the following sections. Then, in §1.5, classifier systems, which are hybrids of the two basic methods, are introduced.

1.4.1 Method 1: Reinforcement Learning Algorithms

The subfield of AI called Reinforcement Learning is concerned with a certain subset of the algorithms which can be used with the RL paradigm mentioned in §1.2.1. It is important to understand the distinction between the RL learning paradigm – a class of learning tasks – and so-called *RL algorithms*, one class of algorithms suited for such tasks. The distinction matters because RL algorithms are not the only methods applicable to these tasks; for example, the evolutionary algorithms of the following section are equally applicable.

RL algorithms learn policies by learning value functions (§E.4.1), which are estimates of the (long-term) utility of components of the policy. In short, RL algorithms iteratively evaluate the utility of the current policy (learn a value function for it) and then derive a better policy based on the value function (§F.3). RL algorithms are introduced in Appendices E and F.

RL algorithms do not address the question of generalisation directly, but often make use of function approximators to provide generalisation. For example, a value function is often stored (approximated) using a neural network, rather than a look-up table. We will make frequent references to Q-learning ([287, 266], §E.4.5), the best-known RL algorithm.

1.4.2 Method 2: Evolutionary Algorithms

The subfield of AI called Evolutionary Computation (EC), introduced in Appendix G, is applicable to both the problem of learning a policy and the problem of generalisation. EC comprises many forms of Evolutionary Algorithms (EAs), all of which, inspired by evolution in nature, employ simulated evolution of populations of candidate solutions. In short, this involves iteratively evaluating the current generation of solutions and producing the next generation by means of probabilistic transformations and selective pressure towards fitter members. EAs can be thought of as stochastic population-based search methods.

1.5 Learning Classifier Systems

Classifier systems originated and have been developed primarily within the field of Genetic Algorithms (see §G.2, and, e.g., [120, 95]), one of the stochastic search methods studied in the field of Evolutionary Computation. Classifier systems are traditionally hybrid Evolutionary/Reinforcement Learning

systems, although some are Supervised Learning systems (e.g., [30]), or use non-evolutionary means of generating rules (e.g., [260]). Although such systems are significant, this work will focus on systems which are hybrids of EC and RL algorithms, and which learn from rewards.

1.5.1 The Tripartite LCS Structure

To return to the two challenges of §1.3, the questions of how rules are evaluated and generated are addressed, respectively, by a classifier system's *credit assignment system* and *rule discovery system*.

Rule Discovery System

The rule discovery system almost always consists primarily of a Genetic Algorithm, which may be supplemented by other mechanisms (see §2.3.6).

Credit Assignment System

A great many credit assignment algorithms exist, and different types will be briefly outlined in §1.5.3. Whatever method is used, the role of the credit assignment system is to observe the use of the rules as the system interacts with its problem environment, and the feedback the system receives from the environment, and update a numerical estimate of each rule's utility called its *strength*. (In the classifier systems for RL in which we are interested, feedback consists of rewards, that is, numbers.)

Production System

In addition to the two systems mentioned above, a classifier systems needs machinery which applies the right rules at the right time, that is, it is responsible for finding the rules which apply, or *match*, the current state of the problem. This machinery constitutes the third major component of a classifier system, the *production system*. We will see in more detail how these systems fit together to produce a working classifier systems in Chapter 2, and will reconsider how they fit together in Chapters 3 and 6.

1.5.2 LCS = Policy Learning + Generalisation

It is worth emphasising that classifier systems intrinsically address both the problem of learning policies and the problem of generalisation. The inherent capacity to generalise is one of the main features which distinguish LCS from other approaches to RL, and a motivation for interest in LCS. Of course other approaches to generalisation in RL exist; the most common is to use a neural network to approximate the value function of, e.g., a tabular Q-learner. In

this case, generalisation is achieved by combining tabular Q-learning with a function approximator. The two processes are distinct. In a classifier system, however, we cannot separate generalisation and policy learning; a classifier system is a combination of both, and if either is missing we do not have a classifier system.

Although the potential to generalise is one reason for interest in LCS, it also makes the problems faced by classifier systems and study of them more difficult (§7.5.2).

1.5.3 Credit Assignment in Classifier Systems

In sequential tasks, credit assignment is normally performed by some form of Temporal Difference algorithm (see [266], §E.4.5), from the field of Reinforcement Learning. Various versions of the bucket brigade algorithm (see [127, 95]) have most often been used, although Q-learning-like updates have become popular in recent years (see [297, 298], §E.4.5).

However, not all LCS use Temporal Difference algorithms; some have used what are referred to as epochal schemes in the LCS literature and Monte Carlo updates (§E.4.4) in the RL literature (following the usage of Sutton and Barto [266]). Such LCS include CS-1 [128] and RUDI [101]. Monte Carlo updates have two disadvantages compared to Temporal Difference updates. First, it is difficult to apply Monte Carlo updates in tasks which do not divide naturally into episodes, that is, which periodically terminate and restart. Such tasks are termed episodic, as opposed to continuing tasks which do not terminate. (See §C.3.3.) Second, Temporal Difference methods take advantage of the sequential structure of a task in a way which Monte Carlo methods do not, and so often outperform them. On the other hand, Monte Carlo methods have some advantage on tasks which are not well-modelled as Markov processes (see §C.5). There is, however, relatively little known about how the two compare [266].

Some of the different types of credit assignment schemes are illustrated in Figure 1.1, which includes strength-based and accuracy-based forms of Temporal Difference schemes, to be introduced in the following section. This work is primarily concerned with strength and accuracy-based schemes, and has little to say about Monte Carlo updates.

1.5.4 Strength and Accuracy-based Classifier Systems

It is difficult to generate new rules efficiently without being able to determine the quality of existing rules, meaning rule evaluation is an issue in rule discovery. For example, the typical approach of using a genetic algorithm requires an estimate of the value of each rule called its *fitness*.

Since rules are evaluated by the credit assignment system for conflict resolution, an obvious approach is to use the same evaluation in rule discovery,

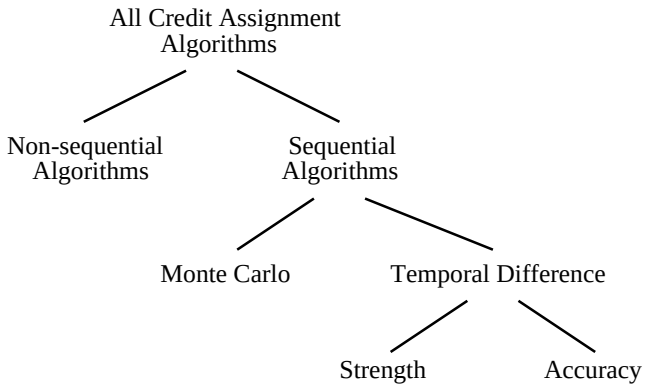

Fig. 1.1. Some types of credit assignment schemes.

and this is just the approach used by *strength-based* classifier systems. That is, these systems use a rule's strength in both action selection and rule discovery.

In contrast, in Wilson's XCS classifier system [298], a rule's strength is used only in action selection. For reproduction, the utility (fitness) of a rule is a different value, although it is ultimately derived from the strength of a rule. More specifically, in XCS, a rule's fitness is a function of the accuracy with which it predicts the reward it will receive, which is why XCS is called an *accuracy-based* classifier system. The difference is illustrated in Figure 1.2, in which arrows indicate the flow of information from its source (rewards) to its use in the processes of action selection and rule discovery. We will see that this simple difference has profound implications for the system. We will take XCS as a representative of a class of related possible accuracy-based systems, and, where possible, consider the class of such systems and not just XCS.

A major aim of this work is to explain the differences between the older strength-based LCS and the newer XCS, and the significance of these differences. We will make extensive comparisons between strength and accuracy-based systems, both theoretically and empirically. In order to do so, a system called SB–XCS (Strength-Based XCS) will be introduced in Chapter 2.

1.6 About the Book

This work arises in response to a number of questions:

- How do strength and accuracy compare?
- Is a classifier system GA-based or RL-based?
- How do classifier systems relate to Q-learning?
- How do different types of classifier system relate to each other?

The goals of this work, then, are to compare two broad types of classifier system: strength-based and accuracy-based, and to improve our understanding

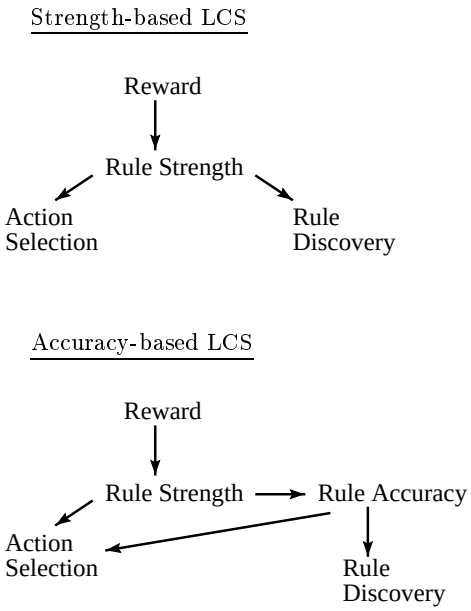

Fig. 1.2. How strength and accuracy-based systems use rewards to weight rules in action selection and rule discovery.

how different learning systems relate to each other, that is, how they 'fit together'. The following section explains the need for comparison between strength and accuracy, while §1.6.2 explains the confusion over the question of whether LCS are GA-based or RL-based. Finally, §1.6.3 casts this work as an exploration of design space, and explains the necessity of this kind of work.

1.6.1 Why Compare Strength and Accuracy?

The creator of XCS, Stewart W. Wilson, motivates the switch to accuracy-based fitness in [298], but this is the only discussion in the literature and a comparison of the two approaches has been lacking. A better understanding of the two approaches is important for a number of reasons.

- The question of how to calculate rule fitness has been the subject of much study over the years. Fitness definition is a fundamental issue since a classifier system will not adapt well if fitness is not defined in a suitable way. Thus, understanding of how strength and accuracy compare is important for the development of the field of classifier systems.
- There is evidence that traditional strength-based fitness is unsuitable for some sequential tasks [298, 56] (but see [48]).
- There is a growing body of evidence that accuracy-based fitness *is* suitable for sequential tasks (e.g., [298, 166, 11, 12, 13, 14]).

- XCS has generated considerable interest and has become a major focus of classifier systems research [150].
- It has been suggested that XCS's complete mapping from inputs and actions to reward predictions is advantageous [298].
- Later in this work it will be suggested that XCS also has important advantages in handling the explore/exploit dilemma (§C.2) and value propagation (§3.7.2) in reinforcement learning, and what we have called strong and fit overgeneral rules (see Chapter 5).
- It has been suggested that accuracy-based fitness shows better generalisation than strength (and consequently requires smaller population sizes) [298, 147, 304].
- However, it has also been suggested that accuracy may require *larger* population sizes than strength (see Chapter 3).

Most of these points suggest an advantage of accuracy-based fitness, and a comparison is needed to ascertain whether this important new direction in LCS research lives up to these expectations.

1.6.2 Are LCS Based on Evolutionary Computation or Reinforcement Learning?

What a classifier system is seems contentious, to the extent that discussion of this issue dominated the First International Workshop on Learning Classifier Systems (IWLCS-92). As Robert Smith, paraphrasing Lashon Booker, reported:

> The LCS is usually described as a *method*: a set of algorithmic details that define a way to solve a class of problems. However, in many ways the LCS is more of an *approach*: a set of conceptual details that define a certain direction for developing methods. Therefore, the defining issues for the LCS are not necessarily algorithmic, but conceptual. The central problem addressed by the workshop's discussions was to clarify these defining, conceptual issues. [243] p. 2.

That conceptual issues remain a concern for classifier systems is indicated by the inclusion of a series of 11 short essays under the title "What is a Learning Classifier System?" in a recent publication [126]. Although I was unaware of this diversity of opinion when I first began working with them in 1996, I soon became concerned that I was unsure of the difference between a genetic algorithm and a classifier system. I had seen the LCS described as a combination of a production system, rule discovery system and credit assignment system. I reasoned that since the rule discovery system typically *is* a genetic algorithm, the LCS must be something more, since it has two additional components. However, I decided the credit assignment system was just what we call the fitness function of a genetic algorithm. Granted, credit assignment in an LCS was more complex than the examples of function optimisation with a

GA which I had seen, but it was still a kind of fitness function. This left the production system as the real difference between the LCS and GA. But since the production system is conceptually straightforward – its task is simply to apply the rules when appropriate – a classifier system seemed to be just a way of applying a GA to certain kinds of problems. Certainly we need to wrap the GA up with a little machinery (the production system and a special kind of fitness function) to interface it with the problem, and perhaps the GA needs a little help in the form of operators like covering (see, e.g., [121, 32, 298], §2.3.6.2), but the LCS seemed to be essentially a GA.[1]

This is a view which I still think is consistent with Holland's intentions, and those of many others. Classifier systems have, after all, been described as *Genetics-Based* Machine Learning (GBML) systems [95].

Less-genetic Classifier Systems

The view of an LCS as essentially a GA is somewhat extreme, and other less extreme views exist. In addition to the GA, a classifier system may contain rule discovery mechanisms such as covering, triggered chaining [214], bridging [123, 211], and corporate linkage [315, 274]. In such systems the GA is just one component of the rule discovery system, although perhaps an important one. However, some LCS emphasise the use of non-genetic operators more heavily than others, and in some cases the GA is even considered a 'background' operator [14].

Non-genetic Classifier Systems

That an LCS is essentially a GA is flatly contradicted by the considerable recent work on LCS which use alternative rule discovery systems. In hindsight, there seems no justification for insisting on the use of GAs as opposed to other evolutionary algorithms. Alternatives were suggested some time ago [297, 278, 279, 244] and some recent work has indeed used Genetic Programming rather than Genetic Algorithms [165, 4]. What's more, a significant amount of recent work has been on systems which contain no evolutionary algorithms [255, 257, 256, 258, 259, 49, 50, 260, 261, 262, 52]. If we accept such systems as classifier systems (as is the norm, e.g., work on such systems has appeared at the International Workshop on Learning Classifier Systems – IWLCS), we are dealing with a much broader concept than that of a GA and some wrapping. Unfortunately, discussion of this trend lies outside the scope of this book.

Linking Classifier Systems and Mainstream RL

A second trend which breaks from the view of an LCS as a GA-based system is that which seeks to link LCS and mainstream reinforcement learning. RL has

[1] A more detailed account of the differences between the two would be desirable, but must be deferred to another work.

made great strides since the introduction of LCS, and it is clear that (most) LCS are RL systems, that they address many of the same issues addressed by other RL systems, and that there is much to be gained from integrating LCS with mainstream RL. The need to bridge LCS and mainstream RL appeared to be the consensus during the discussion at IWLCS-99.

The GA-view and RL-view

This leaves us with two contradictory views of what a classifier system really is, what we might call the *GA-view* – that the LCS is essentially the application of a GA to a problem – and the *RL-view*; that the LCS is a kind of RL system, i.e., a Q-learning-like system in which the GA is (or may be) a component, but in which many of the interesting issues are to do with credit assignment. The two views place different emphasis on different subsystems: according to the GA-view, the GA, and issues relating to it, are of primary importance, while the RL-view places greater importance on credit assignment. The existence of two alternative views begs an important question: does an LCS solve problems using evolutionary means, or does it solve them in the way non-evolutionary RL systems do?

One aim of this work is to recognise and publicise the existence of these alternative views, since they seem under-recognised, particularly in the literature. Another aim is to clarify these views, and to justify the RL-view of (some) LCS. Significantly, the RL-view focuses on XCS, which, it will be argued, differs fundamentally from Holland's LCS, to the extent that it more closely resembles mainstream RL systems, such as tabular or neural network-based Q-learners.[2]

In order to make this RL-view clear, and accessible to the many in the LCS community who are not well versed in RL, this work goes to considerable lengths introducing RL in Appendices C, E and F. Those familiar with RL may wish to skip over these sections, but they are highly recommended to those unfamiliar with the subject. Without a good, basic understanding of RL, the distinction between the GA and RL views of LCS is likely to be unclear. What's more, the future of LCS research (for sequential tasks) seems heavily grounded in this view; the most important issues to be addressed in LCS research are those which are and will be addressed in RL. This is not to marginalise evolutionary approaches to RL, but to say that they too will benefit from understanding of non-evolutionary approaches. The text by Sutton and Barto [266] offers a far more complete introduction to RL, and should be required reading for anyone wishing to apply LCS to RL problems.

[2] Despite this, XCS originated and has been studied exclusively within the LCS community, and is by far most strongly integrated with the LCS literature. Much better integration with mainstream RL awaits.

1.6.3 Moving in Design Space

This section briefly discusses ways in which progress can be made in a scientific field, and the contribution this book makes to the study of LCS.

The Design-based Approach

The literature contains a great number of designs for classifier systems, and will surely see new ones introduced in the future. One can think of these systems as points in the space of possible designs – design space – and the space of requirements for our designs – niche space. These are concepts from the *design-based* approach, which involves taking the role of an engineer who is trying to design a system that meets certain requirements, and is inspired by software engineering and conceptual analysis in philosophy [237, 238, 239, 240]. It involves analysing alternative sets of requirements, designs and implementations in an attempt to establish the nature of their relationships. It allows a high level functional comparison of systems, both natural and artificial, despite differences in origin or implementation. This comparison seeks to identify which aspects of a system are essential for given functions and which are not. Note that this approach does not require a full understanding of the requirements or the available tools at the outset, nor does it assume that there is a single correct design to be found.

The succession of designs for classifier systems can be seen as a search in design space, a search for optimised designs. How does search in design space progress? Ideally, one would make as big a jump as necessary to reach, all at once, an optimum design for one's requirements. But even if the optimum design was found immediately, how would one know one had found the optimum? Without some formal method which can be used to prove the optimality of a design, one can only continue to search for better designs (i.e., consider other designs). (In the study of classifier systems (or Artificial Intelligence more generally), the requirements of our systems are not fully understood, and so the set of requirements an optimal design would meet is not defined. Chapter 4, however, does at least address the issue of the requirements of a classifier system.)

An Analogy with Evolutionary Computation

Search in design space can be thought of in terms of the familiar idea of an evolutionary algorithm moving across a fitness landscape (§G.3). Movement consists of iteratively generating and evaluating new designs. Because a field consists of a population of researchers, many of them generating and evaluating different designs, a community of researchers conducts a parallel, interacting search – effectively an evolutionary algorithm.

Interleaving Analysis and Invention

How well must one evaluate a design before one can move on? There is a form of explore/exploit tradeoff (§C.2) at work; the better one understands the current area of design space, the better one can direct oneself to a preferable region. The cost of gaining deeper understanding is time and effort. As with other explore/exploit problems, the optimal tradeoff between evaluating designs and generating new ones is difficult to achieve.

In Evolutionary Computation one typically only needs to know the approximate fitness of an individual to plausibly generate improved ones. Similarly, in searching design space, one may only need to evaluate a design incompletely before moving on to consider others. For example, if a design fails to meet a vital requirement, it can be rejected without exploring how it meets other requirements.

However, in cases where designs are not evaluated deeply, one is likely to make smaller improvements on them than one might otherwise. That is, in making an informed jump in design space, the size of jump which can be made is limited by how well one understands that space; limited understanding restricts the points to which one can jump. Small jumps may consign us to hill-climbing; approaching a locally optimal design. Bigger jumps can escape local optima, but making bigger jumps – in the direction intended – requires greater understanding of the surrounding space.

XCS as a Jump in Design Space

XCS is a big jump in the design space of classifier systems; even the shift to accuracy-based fitness by itself constitutes a big jump. Though the differences between the specification of XCS and its strength-based twin SB–XCS are quite minor, the resulting systems operate in a very different way. (But then changing a few genes in an animal can have a huge effect.) In fact, it is argued later (Chapter 6) that XCS has more in common with tabular Q-learners than with older classifier systems, or even with its twin SB–XCS.

The Contribution of this Book

So far, XCS has been shown to be worthy of further consideration. However, it is not yet well understood, and neither are earlier classifier systems. This work addresses the difficult problem of showing that strength-based LCS are indeed unsuitable for many tasks, and why this is so. We will see considerable evidence for this argument, particularly in §2.5 and Chapter 5.

I have pursued the study of strength and accuracy presented here because I believed it was the best way to make progress with classifier systems. Rather than extend XCS, or create a new system, I wanted to understand – in detail – the differences between XCS and other LCS, and between LCS and other reinforcement learning systems.

1.7 Structure of the Book

- Chapter 2 begins with a discussion of the range of classifier systems in the literature, and the difficulties of classifying them given our limited understanding of how they compare. It then reviews representations used by classifier systems, in particular the standard ternary language. Finally, it introduces two systems, XCS and SB–XCS, in detail and presents an initial comparison of them.
- Chapter 3 studies the differences between XCS and SB–XCS in greater detail, and considers rationales for why they should adapt to a task. It then examines representational differences between the two, and evaluates the alternative representations.
- Chapter 4 considers what features the representation of a solution should have, and examines metrics for them.
- Chapter 5 analyses what types of rules are possible and under what conditions they can be expected to occur. Competition between selfish rules is shown to produce various forms of rules which are detrimental to the performance of the system as a whole, namely overgeneral, strong overgeneral and fit overgeneral rules. It is shown that XCS does not produce such rules in the circumstances in which SB–XCS does. However, it is also shown that there are circumstances in which XCS will produce such rules. The prospects for adaptation of the two systems to sequential and non-sequential tasks are evaluated.
- Chapter 6 considers the relationship between classifier systems and Q-learning, and revisits the issue of the GA-view and RL-view from §1.6.2.
- Chapter 7, the conclusion, outlines a model of the capacities of various types of classifier systems, lists the contributions of the book, discusses open problems and future work, and ends with some remarks on the history and state of the field.
- Appendix A examines XCS with and without the use of macroclassifiers (§2.3.2) and finds in favour of their use.
- Appendix B walks through a step of the XCS algorithm in order to illustrate the process.
- Appendices C and D introduce the Reinforcement Learning and Generalisation problems respectively.
- Appendix E introduces methods for solving the prediction problem, while F introduces RL methods which are based on policy improvement and Appendix G introduces Evolutionary Computation.
- Appendix H points out that Wilson developed a version of the Sarsa update (§E.4.5) contemporaneously with Rummery and Niranjan.
- Finally, Appendix I lists the notation used in this work.

How to Read this Book

Readers wishing to cover any of the introductions to Reinforcement Learning problems (Appendix C), Generalisation problems (Appendix D), prediction methods (Appendix E), RL algorithms (Appendix F) and Evolutionary Computation (Appendix G) may wish to do so before proceeding to Chapter 2. The material in these Appendices should help clarify Chapters 2–7, although familiarity with this material should not be strictly necessary.