Learning Classifier Systems: Introducing the User-friendly Textbook

Will Browne¹ & Ryan Urbanowicz²
1. Evolutionary Computation Research Group, Victoria University of Wellington, NZ
Will.Browne@vuw.ac.nz
2. Dartmouth College, USA
RyanUrbanowicz@gmail.com

http://www.sigevo.org/gecco-2013/

Copyright is held by the author/owner(s).

Course Agenda
- Introduction
- How LCSs map a problem
  - Demo of 'Classifiers'
- How LCS learn a better map
  - Demo of the main LCSs evolutionary cycle
- Organisation of a LCS
  - Demo of different concepts within LCSs
- Applications of LCSs
  - Demo of different types of LCSs
- Overview, Questions & Discussion

Guides
- Will N. Browne's main area of research is Artificial Cognitive Systems. He has served as Co-track chair for Genetics-Based Machine Learning in GECCO 2011 and 2012 [plus organizing committee of the International Workshop on Learning Classifier Systems (IWLCS) from 2009-2010]. Editor in chief for the Australasian Conference on Robotics and Automation 2012 and organized the Cognitive Robotics Intelligence and Control COGRIC, EPSRC (UK)/NSF (USA)
- Ryan Urbanowicz holds a Ph.D in genetics from Dartmouth College, and both a M.Eng. and B.Eng. in agricultural and biological engineering from Cornell University. His current research focuses on the development of machine learning strategies for feature selection, modeling, classification, and data mining in studies of common complex human disease. He served on the IWLCS organizing committee from 2010-2012 and is returning as an organizer from 2012-2014.

Introduction
- This tutorial will introduce the concept of Learning Classifier Systems (LCS)
- User-friendly in order to allow
  - Graduate students,
  - EC researchers and
  - Industry/business practitioners
who want to get up to speed with the field an easy path into using LCS to solve their complex problems.
Introduction

“LCS are a quagmire - a glorious, wondrous and inventing quagmire, but a quagmire nonetheless” D. Goldberg ’82

Not anymore!

- 30+ years research on LCS has clarified understanding, produced algorithmic descriptions, determined 'sweet spots' for parameters and delivered understandable 'out of the box' code
- This tutorial/book offers a boardwalk through the swamp

Deliverables:
- This tutorial offers a user-friendly guide so that you will be able to proficiently implement LCS.
- It will be through explanation based on the
  - slides accompanying the book,
  - examples supported by the Python code and
  - insight/narrative from the two authors.
- Tutorial participants will have online access to the textbook, slides and code to work through the examples.

By The Way

- “Learning Classifier Systems” is an odd name!
- There are many Artificial Intelligence systems that learn to classifier (such as Decision Trees) that are not Learning Classifier Systems
- Learning Classifier System, abbreviated to LCS, refers to a singular/specific system, whereas Learning Classifier Systems (LCSs) refers to multiple systems or the field.
- Genetics-Based Machine Learning (GBML) is more accurate, but still not completely precise (e.g. Artificial Immune Systems are GBML, but not LCSs).
- Please accept the limitations in the name and let’s explore the the concepts that underline LCSs.

Wondrous as:
Learning Classifier Systems combine the global search of Evolutionary Algorithms with the local optimisation of Reinforcement Learning to address classification and regression problems.

- The knowledge extracted though interacting with data or embedded in an environment is human readable.
- ‘Inventing’ as LCS' flexible nature allows application to many domains with many types of feedback on solution progress.
- But ‘swampy’ as an LCS is not a one line algorithm with separable methods and easily tuned parameters.
LCSs [History 1 of 1]:

- Learning Classifier Systems are one of the earliest artificial cognitive systems.
- The early work was ambitious and broad. This has led to many paths being taken to develop the concept over the next 30 years.
- Coupling this with the fact that replicating cognition itself is a difficult problem has led to the field being affectionately termed ‘a quagmire’ and a lack of widespread adoption.
- Early 90s simplified and essential ideas, then understanding and adaption to real-world problems.

Assumptions for AI:

- In order for artificial learning to occur data containing the patterns to learn is needed.
- This can be through recorded past experiences or interactive with current events.
- Learning is often in the harness of a cognitive system as input data, representation, reasoning, learning and output are needed to interact with an environment.

If there are no clear patterns in the data, then LCSs will not learn.

Why learn using AI:

- Learning is valued by humans as it enhances our abilities to solve problems and adapt to our environment.
- Much work in research fields, such as education, psychology and neuroscience, has been conducted into how humans learn.
- With the advent of computers, humans have been interested in seeing how artificial ‘agents’ could learn. Either learning to
  - solve problems of value that humans find difficult to solve
  - for the curiosity of how learning can be achieved.

if ... then ...

The ‘if ... then … ’ statement format essential to LCSs.

- Similarities to production rules in computer science, so the name ‘rule’ is used.
- The learned patterns are represented in the form of ‘if <this> then <that>’ rules.
- Whether
  - <conditions> <action>,
  - <state> <action>,
  - <features> <class> and many other

For convenience we will refer to conditions and actions
Worth of a Rule

An ‘if ... then ... ’ rule may be valid syntactically, but we need to verify its worth

- A valid rule can quite easily encode meaningless relationships and information.
- Interestingly, the majority of valid rules are likely to contain incorrect information.
- In LCSs the worth of a rule is termed ‘fitness’, due to analogies of biological fitness.

Fitness

Fitness is central to the operation of LCSs. It can relate to
- External effects (e.g. the prediction of feedback from the environment)
  and/or
- Internal effects (e.g. overall contribution of the rule to the system)
- Instantaneous, filtered or long-term values may be used.

Overview of LCSs’ purpose

INTERACTION WITH ENVIRONMENT
HAVE: DATA
‘IF... THEN...’ RULES
WANT: CLASSIFIERS

LCS as Parametric Models

- LCS are a family of methods for handling unsupervised learning, supervised learning and sequential decision tasks by decomposing larger problem spaces into easy-to-handle sub-problems.
- Viewpoints
  - evolutionary computation,
  - probabilistic model-based approach
- Defining question "What is an LCS supposed to learn?"
  An underlying probabilistic model
  Drugowitsch ’08
LCSs as Map Generators

The intention is to form a map of the problem space.

Environmental interaction

**Supervised learning:** The environment contains a teacher that (directly or indirectly) provides the correct response for certain environmental states as a training signal for the learning signal.

**Unsupervised learning:** The learning system has an internally defined teacher with a prescribed goal that does not need utility feedback of any kind.

**Reinforcement learning:** The environment does not directly indicate what the correct response should have been. Instead, it only provides reward or punishment to indicate the utility of actions that were actually taken by the system.

[Sutton and Barto 98]
Utilisation of LCSs

- Perpetually novel events accompanied by large amounts of noisy or irrelevant data.
- Continual, often real-time, requirements for actions.
- Implicitly or inexactley defined goals.
- Sparse payoff or reinforcement obtainable only through long action sequences.

[Booker 89]

- Main aspects found in problem domains:
  - Multimodal
  - Lack of Separation
  - High Dimensionality
  - Epistasis

Fitness

- Fitness describes the worth of a rule.
  - embody the past success of the rule
  - indicate the quality of the knowledge held
  - Indicate the utility of the rule to the system

- There are many ways to calculate fitness

Fitness = \frac{\text{number of correct classifications}}{\text{experience}}

Rule + Statistics = Classifier

- Classifiers are not just rules as they contain additional information:
  - 'if < conditions > then < action >' with statistics

  Prediction \( p \) Error \( \epsilon \) Accuracy \( \kappa \) Fitness \( F \)

- Note: high predicting rules may not be fit if they are not consistent

- Many other additional statistics possible:
  - Experience, Number of offspring, Generality, Numerosity

Cooperation

- One rule models a distinct part of the data (a rule covers a single niche in the domain).
- If there was only one niche in the domain, then only one rule would be needed.
- Domains of interest have multiple parts that require modelling with different rules.
- LCSs must learn a set of rules
- The rules within an LCS are termed the population, which is given the symbol \([P]\) - the set of all rules in the population.

- The rules within a population cooperate to map the domain
Competition

- Ideally, there would only be one unique and correct rule for each niche
- Number of rules would equal number of niches
- No prior knowledge, so each rule must be learnt.
- LCSs allow multiple, slightly different rules per niche
- Multiple hypotheses are available to find the optimum rule
- Each rule ‘covers’, i.e. describes, its part of the search space.
- The rules within a niche compete to map the domain.

Learning

- ‘learning’ has a very useful definition
  “Learning is constructing or modifying representations of what is being experienced” Michalski et al., 86.
- LCSs need to experience the domain in order to learn.
  - embodied in an actual robot or
  - ‘virtual’ as in a software program receiving data
- Noise and dynamics within the data may impact on learning ability, but LCSs have shown robustness
- LCSs construct rules or modify existing rules in order to learn

Cooperation & Competition

Grey represent ideal niche.

Which is the most useful plausible rule (stripes)?

Search space

- A major factor in determining the difficulty/likelihood of success.
- Every valid rule can be thought of a candidate solution, i.e. rules that satisfy problem constraints and encoding
- Note, we do not yet know if it is a good solution to the problem or not.
- Each candidate solution is a member of the set of possible solutions.
- The space of all candidate solutions is termed the ‘search space’ [alternative names, such as feasible set, feasible region and solution space exist, but are rarely used in LCS literature].
- The size of the search space is determined by both the encoding of the LCS itself and the problem itself.
Representation

- Environment input must be encoded
- LCSs can use multiple representation schemes.
  - Suited to binary input or
  - Suited to real-valued inputs and so forth...
- Consider representing the hours in a day in four bit binary
- ... the genotype 00011 is expressed as the phenotype 7
- The distance between similar genotypes (and their expressed phenotypes) in a search space is an important consideration when deciding upon the representation

Don't Care

- A condition that we don't care about is given the symbol '#'

For example,

- 101:1 - the Boolean states 'on off on' has action 'on'
- 001:1 - the Boolean states 'off off on' has action 'on'

Can be encoded as

- #01:1 - the Boolean states '. off on' has action 'on'

- The ternary alphabet in the Classifier matches binary input
- In many instances, # acts as an OR function on \{0,1\}

Redundancy, irrelevance and compactness

- A search space has a number of dimensions.
- A single condition encodes a single dimension (feature).
- If the problem has multiple dimensions (features) then the LCS will need the corresponding number of conditions.

- The division between the conditions is implicit within a classifier. For example, the condition string: ‘100110’
  - 1x6-bit number, 2x3-bit numbers or 6xBoolean state
- Not all conditions are useful in the map

LCS Recap

- A basic LCS consists of
  - data is clustered in a population of classifiers,
  - a set of classifiers can be interpreted as a model for the data
  - the most appropriate model is selected

[Drugowitsch 08]
Steps to Evolution

procedure evolutionary algorithm
begin
    t ← 0
    initialise P(t)
    evaluate P(t)
    while (not termination-condition) do
        begin
            t ← t + 1
            select P(t) from P(t - 1)
            alter P(t)
            evaluate P(t)
        end
    end
end

LCSs 'Iterative' Cycle

<table>
<thead>
<tr>
<th>State</th>
<th>Initial Rule Base</th>
<th>Training Rule Base</th>
<th>Encoding</th>
<th>Match</th>
<th>Select</th>
<th>Effect</th>
<th>Credit</th>
<th>Actions</th>
<th>Reward</th>
<th>Reward?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LCSs walk through

<table>
<thead>
<tr>
<th>Conditions</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>[M]</td>
</tr>
<tr>
<td>#011: 01</td>
<td>43</td>
</tr>
<tr>
<td>11##: 00</td>
<td>.01</td>
</tr>
<tr>
<td>#0#1: 11</td>
<td>99</td>
</tr>
<tr>
<td>001#: 01</td>
<td>43</td>
</tr>
<tr>
<td>#0#1: 11</td>
<td>99</td>
</tr>
<tr>
<td>1#01: 10</td>
<td>24</td>
</tr>
</tbody>
</table>

| States: e.g. 0011:01, 1101:00, ... |

Initialisation of an LCS

- A LCS's initial population can be:
  - Empty, Partially Full or Full

- Partially Full or Full can be:
  - Seeded (used to domain knowledge)
  - Random (avoids human bias)

- Empty
  - Utilise environmental messages (cover method)
  - Good for sparse domains and unbalanced data
  - Nowadays the method of choice, especially supervised learning.
Cover method

- Verb 'to Cover', or noun 'Coverage'
- Rule creation:
  - **Condition**: Generalisation of the environmental instance
  - **Action**: Known (supervised learning) or Random (reinforcement learning)

Environmental input: 101101 with corresponding action 1 with probability of generalisation (P#) of 0.33

Rule $a_1$ 1##101:1 + statistics

Match

- Do any of the rules match the input?
  - **Match method**: for example, new input 111101 also action 1.
    - **Specific**: rule conditions exactly match the input
      - 1----- matches Rule $a_1$ 1##101:1
    - **General**: 'don't care' matches all input
      - 1----- matches Rule $a_1$ 1##101:1

Environmental input: 111101 with corresponding action 1

Matches the previously generated rule

Rule $a_1$ 1##101:1 + statistics

All matching rules placed in the match set $[M]$

Explore vs. Exploit

- One of the biggest problems in evolutionary computation
  - When to exploit the knowledge that is being learnt?
  - When to explore to learn new knowledge?
- Annealing schemes must be set a priori without knowledge of the optimum scheme
- Often just a fixed ratio!
Select

Rules in [M] advocate for different actions!
- Consider the statistics of the rules
  - Rule a: \( p = 1000, \ v = 0.1, F = 0.8 \)
  - Rule b: \( p = 800, \ v = 0.4, F = 0.3 \)

- Exploit
  - Greedy (winner takes all) of \( p \)
  - Best average for each action: either \( p \) or \( p \times F \)

- Explore
  - Random action

Evaluate (supervised)

Experience is increased

Accuracy is calculated
\[
\text{acc} = \frac{\text{number of correct classifications}}{\text{experience}}
\]

- Fitness is computed as a function of accuracy:
\[
F = (\text{acc})^{\nu}
\]

Mu \( \nu \) used to separate similar fitness classifiers (e.g. = 10)

Evaluate (reinforcement)

- Recency weighted update

- Widrow-Hoff update: learning rate \( \beta \)
  \[
  \text{value}_{\text{new}} = \text{value} + \beta \times (\text{signal} - \text{value})
  \]

- Filters the 'noise' in the reward signal
  \( \beta = 1 \) the new value is signal, \( \beta = 0 \) then old value kept

- Classifier considered accurate
  if error < tolerance, otherwise scaled.

- Accuracy relative to action set

- Fitness based on relative accuracy
\[
\begin{align*}
p & \leftarrow p + \beta(R - p), \\
v & \leftarrow v + \beta(R - p - v), \\
\kappa & = \begin{cases} 
1 & \text{if } v < v_0, \\
(\varepsilon / v_0)^{-\nu} & \text{otherwise}, 
\end{cases} \\
\kappa' & = \sum_{i=1}^{\kappa} \kappa_i, \\
F & \leftarrow F + \beta(\kappa' - F)
\end{align*}
\]
Select

- Rules differentiated based on fitness value
- In roulette wheel selection often raise to a power (which needs setting)
- Setting the power too high leads to local optimum at the start of training

Rule discovery

- **When to learn**
  - Too frequent: unsettled [P]
  - Too infrequent: inefficient training
  - rank based or relative rating

- **What to learn**
  - Most frequent niches or
  - Underrepresented niches

- **How much to learn**
  - How many good rules to keep (elitism)
  - Size of niche

Select

- Rules differentiated based on fitness rank

<table>
<thead>
<tr>
<th>Action</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

- Rank each action in order of ‘prediction’
- Chosen number of actions selected randomly
- Best action is effected
- This is rarely used to select for effect
- Often used for select for reproduction

Set-based Niching

- Update and creation of classifiers may occur in one of three ways:

  1. Panmictic [P] ‘throughout the population’
     [SCS Goldberg 89]

  2. Match Set [M] restricted to the match set only
     [ZCS Wilson 94]

  3. Action Set [A] restricted to the action set only
     [XCS Wilson 95]
Rule Discovery

Needed to create hypothesised better rules from existing rules & genetic material:

- Genetic algorithm
  - Original and most common method
  - Similar to the niched GAs method
  - Well studied (if not well understood theory)
  - Stochastic process

- Estimation of distribution algorithms
  - Sample the probability distribution, rather than mutation or crossover to create new rules
  - Exploits genetic material

- Bayesian optimisation algorithm
  - Use Bayesian networks
  - Model-based learning

Use any directed method to find new rules!

Mating – Crossover

GA: \[ r_1 = 00010001 \]
\[ r_2 = 01110001 \]

Set crossover point

GA: \[ r_1 = 00010001 \]
\[ r_2 = 01110001 \]

Applying single crossover point

GA: \[ r_1 = 00010001 \]
\[ r_2 = 01110001 \]

\[ c_1 = 01010001 \]
\[ c_2 = 00110001 \]

Crossover Complete

Many variations of crossover possible:
- Two point crossover
- Multipoint crossover
- X-dimensional crossover

Mating – Mutation

GA: \[ 00010001 \]

Randomly select bit to mutate

GA: \[ 10010001 \]

Mutation complete

GA: \[ 10010001 \]

Deletion

- No deletion
  - Population grows without bound, which reduces set pressure
  - Waste memory and takes time so not often used

- Panmictic deletion
  - Most common technique based on inverse fitness roulette wheel
  - Complements set pressure

- Genotypic deletion based on generality
  - Adds bias, hard to set up and control

Advanced deletion schemes – see later
Learning Classifier Systems

Michigan Approach
- Individual is a classifier
- Cooperation and Competition between every classifier
- Population holds all individuals including incorrect, overgenerals, too specific rules

\[ \text{e.g. 3-bit MUX} \]
\[ \begin{array}{ccc}
C & A & F \\
1 & 1 & 50 \\
1 & 1 & 100 \\
11 & 1 & 10 \\
00 & 0 & 89 \\
00 & 0 & 100 \\
\end{array} \]

Pittsburgh Approach
- Individual is a rule set, i.e. multiple rules
- Competition between every rule set to breed (Cooperation is explicit within individual)
- Population holds each unique rule set

\[ \text{e.g. 3-bit MUX} \]
\[ \begin{array}{ccc}
\text{CA} & \text{CA} & \text{CA} & \text{F} \\
11:1 & 1:1 & 00#:0 & 89 \\
11#:1 & 000:0 & 1#1:1 & 45 \\
\end{array} \]

Much debate on best approach!
Rights + Wrongs

- Accuracy measures the inverse of predictive error
- What happens if you are always wrong?
- If you can predict this accurately, then you are 100% accurate!
- In some cases, e.g. MUX, this doubles the rule base

| C | A | P | F |...
|---|---|---|---|---
| 01# | 1 | 100 | 100 |
| 00# | 0 | 100 | 100 |
| 1#1 | 1 | 100 | 100 |
| 1#0 | 0 | 100 | 100 |
| 01# | 0 | 0 | 100 |
| 00# | 1 | 0 | 100 |
| 1#1 | 0 | 0 | 100 |
| 1#0 | 1 | 0 | 100 |

Deferred reward

- Prediction $p$ is updated as follows:

$$p \leftarrow p + \beta [r + \gamma \max P(s',a') - p]$$

where
- $\gamma$ is the discount factor
- $r$ is reward, $\beta$ is learning rate
- $s$ is state, $a$ is action

- Compare this with Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma Q'(s',a') - Q(s,a)]$$

where
- $\alpha$ is learning rate

Path Habits

- Delayed rewards cause problems
- Move NW strengthened early
- Becomes basin of attraction
- Explore does not find E move!

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*</td>
<td>F</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Messy Coding

- Useful for big data, i.e. many features
- Derived from Messy GAs
- Instead of using a don’t care symbol can remove the feature from the condition

11##0:1 shorten to 110:1 with encoding
- Improves transparency, reduces memory and speeds processing
- Used in bio-informatics based LCS, e.g. BioHEL and GAssist by J.Bacardit, and scalable LCS, e.g. XCSCFC
**Choice of Encoding**

Euclidean and Hamming distances alter search space

<table>
<thead>
<tr>
<th>Integer</th>
<th>Binary</th>
<th>Gray</th>
<th>Enumerated</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
<td>00000000</td>
</tr>
<tr>
<td>1</td>
<td>001:1</td>
<td>001:1</td>
<td>0000001:1</td>
</tr>
<tr>
<td>2</td>
<td>010:2</td>
<td>011:1</td>
<td>0000011:1</td>
</tr>
<tr>
<td>3</td>
<td>011:1</td>
<td>010:1</td>
<td>0001111:1</td>
</tr>
<tr>
<td>4</td>
<td>100:3</td>
<td>110:1</td>
<td>0011111:1</td>
</tr>
<tr>
<td>5</td>
<td>101:1</td>
<td>111:1</td>
<td>0011111:1</td>
</tr>
<tr>
<td>6</td>
<td>110:2</td>
<td>101:1</td>
<td>0111111:1</td>
</tr>
<tr>
<td>7</td>
<td>111:1</td>
<td>100:1</td>
<td>1111111:1</td>
</tr>
</tbody>
</table>

How to encode the range: $0 \rightarrow 3$ and $0 \rightarrow 4$ ?

**Integer and Real Encodings**

- Encoding suited to the environmental message
  - using upper and lower bounds:

<table>
<thead>
<tr>
<th>Alphabet</th>
<th>Rule</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>111001 u : 0</td>
<td>11#00#</td>
</tr>
<tr>
<td></td>
<td>110000 l</td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>1.11.10:0:0:0 u : 0</td>
<td>1#000</td>
</tr>
<tr>
<td></td>
<td>0.0:0:0:0:0:0 l</td>
<td></td>
</tr>
</tbody>
</table>

[Could use centre and spread, but this assumes a Gaussian distribution and recombination more difficult to implement]

- For each allele $a$, $lb \leq x \leq ub$, to give match.

- Could use ‘<’ instead of ‘≤’ but LCSs determine the correct bound automatically
  
  e.g. $0 \leq x \leq 5$ is equivalent to $0 \leq x < 5.01$

**How to Match**

Consider the message: 110001

<table>
<thead>
<tr>
<th>Alphabet</th>
<th>Rule</th>
<th>Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ternary</td>
<td>1#1000</td>
<td>1#1000</td>
</tr>
<tr>
<td>Ternary</td>
<td>11#00# l</td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td>111000 u : 0</td>
<td>1#1000</td>
</tr>
<tr>
<td></td>
<td>101000 l</td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td>111001 u : 0</td>
<td>11#00#</td>
</tr>
<tr>
<td></td>
<td>110000 l</td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>1.11.10:0:0:0 u : 0</td>
<td>1#000</td>
</tr>
<tr>
<td></td>
<td>0.0:0:0:0:0:0 l</td>
<td></td>
</tr>
<tr>
<td>Real</td>
<td>1.11.10:0:0:0 u : 0</td>
<td>11#00#</td>
</tr>
<tr>
<td></td>
<td>0.0:0:0:0:0:0 l</td>
<td></td>
</tr>
</tbody>
</table>

**Mutating at the Limits**

- Crossover point can either be
  - between alleles or in the middle of an allele.

- Mutation increases/decreases either or both bounds.
- Repair is occasionally needed to check that: $lb < ub$

- Note that most bounds have a limit, e.g. WBC: $0 \leq a \leq 10$
  - We decide to mutate the lower bound of a ‘#’ $0 \leq x \leq 10$
  - If decrease by 10% of range to -1, repair back to 0.
  - If increase by 10% of range to 1, do not repair as valid!

  *Thus some alphabets have a specificity bias*
Hyper Partitioning

We have a sparse search space with two classes to identify \([0, 1]\). It's real numbered so we decide to use bounds: e.g. \(0 \leq x \leq 10\), which works fine in this case...

We form Hypercubes with the number of dimensions = the number of conditions Approximates actual niches, so Classes 2 & 3 difficult to separate with this encoding, so use Hyperellipsoids

Oblique domains

We have a search space with only two classes to identify: 0 & 1. It's real numbered so we decide to use bounds: e.g. \(0 \leq x \leq 10\)

We form Hypercubes / Hyperrectangles, but these are not often suited to oblique domains Imagine sine wave domains.....

Alternative representations

Many other representations available

- Artificial neural networks
- Fuzzy logic/sets
- Horn clauses and logic
- S-expressions, GP-like trees and code fragments.
  - Is a LCS with S-expressions not just GP? NO!
  - How to tailor functions without introducing bias?
  - How to identify building blocks of Subexpressions?
  - When are two Subexpressions equivalent?
- Is trade-off between reduced problem search space to increased alphabet search space worth it?

Gracefulness

- Need to introduce new rules into population
  - Zero fitness: never selected
  - Full fitness: always selected destabilising existing rules (especially if new rule is poor)
  - Parents average: often unrepresentative
- Moyenne Adaptive Modifiee followed by Widrow-Hoff update
- MAM: Simple time average from start \([1/\beta \text{ iterations}]\)
- WH: Recency weighted average

[Have to set when recency becomes important and how many times steps is recent?]
Overgenerals
[undesired inaccurate classifiers]

- When additional reward offsets any additional penalty
- Strength-based fitness is more prone to overgenerals
- Accuracy-based fitness is less prediction orientated

Want 10011###1:1 get 10011####:1, where 10011###0:0

- Can occur in unbalanced datasets or
- where the error tolerance $\varepsilon_0$ is set too high.

Subsumption deletion

- In sparse environments over-specific rules can take over population
  Want 10011###1:1 get 10011#011:1, 100111111:1, …

- Starvation of generals, so delete specific 'sub-copies'
- Need accurate rules first: how to set level of accuracy
  (often not 100%)
- GA or action set [A] subsumption?
- Effect of numerosity?

Fitness Pressure

- Fitness pressure is fundamental to evolutionary computation: "survival of the fittest"

- Fitter rules assumed to include better genetic material,
- Fitter rules are proportionately more likely to be selected for mating,
- Genetic material hypothesised to improve each generation.

- Fitness measures based on error or accuracy drive the population to rules that don’t make mistakes
- Favours specific rules (cover less domain)
- Fitness measures based on reward trade mistakes for more reward
- Favours general rules (cover more domain)
**Set Pressure**
- Set pressure is related to the opportunity to breed,
- Does not occur in panmictic rule selection
- Need Niching through [M] or [A] rule discovery
- Class imbalance affects set pressure
  - Set pressure is more effective when replacing ‘weaker’ rules
  - Often panmictic deletion, thus one action can replace a different action
  - To prevent an action type disappearing, relative fitness is used (rare rules have high relative fitness and so breed)
- Rules that occur in more sets have more opportunity to be selected from mating
- Favours general rules

**Mutation pressure**
- Genotypically change the specificity-generality balance
- Mutation can
  - Randomise:  
    - Generalise:  
      - $0 \leftarrow 1$ or $\#$
      - $1 \leftarrow 0$ or $\#$
      - $\# \leftarrow 0$ or $1$
    - Specialise:  
      - $0 \leftarrow 0$
      - $1 \leftarrow 1$
      - $\# \leftarrow 0$ or $1$

**Subsumption Pressure**
Subsumption occurs where one rule subsumes another, i.e:
- Rule A subsumes Rule B when they both have the same action, but rule A covers rule B completely

For example:
- Rule A: 11###1:1
- Rule B: 1101#1:1

If rule A is completely accurate ($\epsilon < \epsilon_0$) Then can delete rule B from the population without loss of performance

Favours general, but not over general, rules

**Subsumption Pressure**
Subsumption deletion may occur in two places:
- In the action set
  - Small number of rules to check
  - Check is run often
  - Subsumed rules in population until they occur in an [A]
- In the Rule discovery
  - Large number of rules to check
  - Check is run infrequently
  - Subsumed rules never enter population
- Rules that subsume other rules
  - including copies of themselves (children can be exact copies of their parents)
  - have their numerosity increased.
Numerosity
Numerosity is a useful concept (trick):

- Reduces memory usage
  - Instead of population carrying multiple copies of the same classifier it just carries one copy.
  - Each rule has a numerosity value (initialised as 1)
- Protects rule from deletion
  - Stabilises rule population
- Numerosity is increased by 1
  - When subsumes another rule
  - When RD makes a copy
- Numerosity is decreased by 1
  - Rule is selected for deletion

Deletion pressure
1. Every classifier keeps an estimate of the size of the match sets in which it occurs.
   The estimate is updated every time the classifier takes part in an [M]
   A classifier’s deletion probability is set proportional to the match set size estimate, which tends to make all match sets have about the same size, so that classifier resources are allocated more or less equally to all niches (match sets).
   Then the probability from (1) is multiplied by the mean fitness divided by the classifier's fitness.”

Numerosity affects select and update procedures:

- When calculating the fitness of an action in the select for effect procedure:
  - The fitness used assumes all the classifiers in the population:
    \[ f = \frac{\sum F \times p \times n}{\sum F \times n} \]
    Where n is the numerosity
  - Macroclassifiers: all unique classifiers \( n \geq 1 \)
  - Microclassifiers: all individual classifiers
    (n copies of macroclassifiers)
  - Ratio of macroclassifiers to microclassifiers often used as a measure of training progress

Deletion pressure
- The stronger the deletion bias against low fit classifiers, the more the system will tend to delete useful new classifiers before it realises how good they are.

3. A hybrid in which (1) is used until a classifier has been used on \( n \) trials, after which (2) is used.
   n is called the delay for t3 as it controls how long we delay the application of the low fitness penalty.

   Alternatively, classifiers may be prevented from reproduction until they have experienced a set number of trials.
   - This allows a higher initial fitness to be used, enabling simpler deletion methods to be considered.
   - However, each classifier must record number of its trials and the parent selection method becomes slightly more complex.
XCS
Arguably the most adopted LCS:
- Not an acronym!
  - Although rumoured to be eXtended Classifier System
- Extends basic ZCS
- Niche based fitness to add set pressure
- Accuracy based LCS
- Complete and accurate mapping from inputs and actions to pay off predictions
- Maximally general subject to an accuracy criterion (aided by subsumption)

Complete map
- Should LCS discover:
  - The most optimum action in a niche or
  - The predicted payoff for all actions in a niche
  \[X \times A \Rightarrow P\] (cf Q-Learning)
- The danger with optimum action only:
  - If a suboptimal rule is converged upon ... difficult to discover and switch policy (CF path habits)
- The problem with predicting all actions:
  - Memory and time intensive
  - Identifies and keeps consistently incorrect action (100% accurate prediction) rules
  - Harder to interpret rule base

Classifier parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p)</td>
<td>Prediction (p): Estimates keeps average off the pay-off expected if the classifier matches and its action is taken by the system</td>
</tr>
<tr>
<td>(e)</td>
<td>Prediction error: Estimates the errors made in the predictions</td>
</tr>
<tr>
<td>(F)</td>
<td>Fitness: Denotes the classifier’s fitness</td>
</tr>
<tr>
<td>(\text{exp})</td>
<td>Experience: Counts the number of times since its creation that the classifier has belong to an action set</td>
</tr>
<tr>
<td>(t_s)</td>
<td>Time stamp: Denotes the time-step of the last occurrence of a GA in an action set to which this classifier belonged</td>
</tr>
<tr>
<td>(a_s)</td>
<td>Action set size: Estimates the average size of the action sets this classifier has belonged to</td>
</tr>
<tr>
<td>(n)</td>
<td>Numerosity: Reflects the number of micro-classifiers (ordinary classifiers) this classifier – which is technically called a macroclassifier – represents</td>
</tr>
</tbody>
</table>

Learning parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N)</td>
<td>Maximum size of the population (in micro-classifiers, (N) is the number sum of the classifier numerosities)</td>
</tr>
<tr>
<td>(\beta)</td>
<td>Learning rate for (p, e, F, \text{exp}) and (a_s)</td>
</tr>
<tr>
<td>(\delta, \epsilon, \gamma)</td>
<td>Used in calculating the fitness of a classifier</td>
</tr>
<tr>
<td>(\theta_{\text{del}})</td>
<td>GA threshold: The GA is applied when the average time since the last GA in the list is greater than the threshold.</td>
</tr>
<tr>
<td>(\chi)</td>
<td>Probability of applying crossover in GA</td>
</tr>
<tr>
<td>(\mu)</td>
<td>Probability of mutating an allele in the offspring</td>
</tr>
<tr>
<td>(\theta_{\text{sub}})</td>
<td>Deletion threshold: If the experience of a classifier is greater than (\theta_{\text{sub}},) its fitness may be considered in its probability of deletion</td>
</tr>
<tr>
<td>(\delta)</td>
<td>Mean fitness in ([\text{P}]) below which the fitness of a classifier may be considered in its probability of deletion</td>
</tr>
<tr>
<td>(\theta_{\text{sub}})</td>
<td>Subsumption threshold – the experience of a classifier must be greater than (\theta_{\text{sub}}) in order to be able to subsume another classifier</td>
</tr>
</tbody>
</table>
### Learning parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_p)</td>
<td>Probability of using a # in one attribute in (C) when covering</td>
</tr>
<tr>
<td>(p, \epsilon, F_i)</td>
<td>Initial Values in new classifiers</td>
</tr>
<tr>
<td>(P_{\text{act}})</td>
<td>Probability during action selection of choosing the action uniform randomly</td>
</tr>
<tr>
<td>(\theta_{\text{min}})</td>
<td>Minimal number of actions that must be present in a match set ([M]) or else covering will occur</td>
</tr>
</tbody>
</table>

- \(\text{doGAsumption}\) is a Boolean parameter that specifies if offspring are to be tested for possible logical subsumption by parents.
- \(\text{doActionSetSubsumption}\) is a Boolean parameter that specifies if action sets are to be tested for subsuming classifiers.

### Results Interpretation

<table>
<thead>
<tr>
<th>No.</th>
<th>Condition</th>
<th>Action</th>
<th>(n)</th>
<th>(f)</th>
<th>(e)</th>
<th>(p)</th>
<th>(P_{\text{exp}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 0 1 0</td>
<td>0 2 4 8</td>
<td>0 2 4 8</td>
<td>0 2 4 8</td>
<td>0 2 4 8</td>
<td>0 2 4 8</td>
<td>0 2 4 8</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 0</td>
<td>1 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
</tr>
<tr>
<td>3</td>
<td>0 0 1 0</td>
<td>0 2 4 8</td>
<td>1 2 4 8</td>
<td>1 2 4 8</td>
<td>1 2 4 8</td>
<td>1 2 4 8</td>
<td>1 2 4 8</td>
</tr>
<tr>
<td>4</td>
<td>0 1 0 1</td>
<td>0 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>5</td>
<td>0 1 1 0</td>
<td>0 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>6</td>
<td>0 1 0 1</td>
<td>0 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>7</td>
<td>0 1 0 0</td>
<td>0 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>8</td>
<td>0 0 0 0</td>
<td>1 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
</tr>
<tr>
<td>9</td>
<td>0 0 0 0</td>
<td>1 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
</tr>
<tr>
<td>10</td>
<td>0 0 0 0</td>
<td>1 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
</tr>
<tr>
<td>11</td>
<td>0 0 0 0</td>
<td>1 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
</tr>
<tr>
<td>12</td>
<td>0 0 0 0</td>
<td>1 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
</tr>
<tr>
<td>13</td>
<td>0 0 0 0</td>
<td>1 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
</tr>
<tr>
<td>14</td>
<td>0 0 0 0</td>
<td>1 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
</tr>
<tr>
<td>15</td>
<td>0 0 0 0</td>
<td>1 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
<td>0 2 3 4</td>
</tr>
</tbody>
</table>

Note: 2, 3, 5, 7, 11, 14, 15 incorrect, but accurate, so fit

### XCSF

- **Real valued encoding:**
  - Concatenation of "interval predicates"
  - \(l_i \leq x_i \leq u_i\)
- **Crossover:** Two-point crossover point can be between predicates within an interval predicates
- **Mutation:** addition of an amount \(\pm \text{rand}(m_0)\)
  - returns value from \((0, m_0]\) for lower bound
  - Bounded random amount proved better than fixed amount
- **Repair** used as normal
- **Covering:** lower bound
  - \(l_i = x_i - \text{rand}(r_0)\)
  - Returns value from \([0, r_0)\)
XCSF: Piecewise-Linear Approximation

- **Linear approximation:**
  The action is flexible for each condition range (every time classifier matches it has the same action)

- **Need to approximate the function**
  \[ y = f(x) \]
  With \( h(x) \)
  \[ h(x) = w_0 + w_1x_1 \]

- **Prediction is linear polynomial of the input components**
  Initially approximate a 1-D function:
  - \( w_1 \) slope of a straight line
  - \( w_0 \) with its intercept
  Hyperplane approximation to \( f(x) \)

Cognitive Systems definition

Current robots are poor cognitive systems. Need to improve devices that we use every day and investigate medical benefits.

"Cognitive systems are natural or artificial information processing systems, including those responsible for perception, learning, reasoning and decision-making and for communication and action"

DTI Foresight initiative.

Cognitive LCS

**Induction: Processes of Inference, Learning and Discovery (Computational Models of Cognition and Perception)**

*J. H. Holland, K. J. Holyoak, R. E. Nisbett and P. Thagard*

Two psychologists, a computer scientist, and a philosopher have collaborated to present a framework for understanding processes of inductive reasoning and learning in organisms and machines. Theirs is the first major effort to bring the ideas of several disciplines to bear on a subject that has been a topic of investigation since the time of Socrates. The result is an integrated account that treats problem solving and induction in terms of rulebased mental models.


Anticipatory LCS

- **Anticipations influence cognitive systems and illustrates the use of anticipations for**
  1. Faster reactivity
  2. Adaptive behavior beyond reinforcement learning
  3. Attentional mechanisms
  4. Simulation of other agents
  5. The implementation of a motivational module.

- **A particular evolutionary model learning mechanism, a combination of**
  - a directed specializing mechanism and
  - a genetic generalizing mechanism.

- **Experiments show that anticipatory adaptive behavior can be simulated by exploiting the evolving anticipatory model for even faster model learning, planning applications, and adaptive behavior beyond reinforcement learning.**
LCS generates accurate and general rules covering states, together with a utility of the rules.

Abstraction algorithm generates meta-rules covering the discovered accurate rules.

LCS generates accurate and general rules covering states, together with a utility of the rules.

Abstracted Rules

Abstracted Rules

e.g. 'if side guide setting < width, then poor quality product'

Abstraction checks for patterns in the base rules and crates and abstracted rules for each discovered pattern.

Base rules

e.g. if side-guide-setting = 80, width = 82 then poor quality product
if side-guide-setting = 79, width = 80 then poor quality product

Learning system

Raw Data

e.g. Features:
'side-guide setting', 'width': 'product quality'
78 81: poor
79 80: poor
78 76: good
...

Connect4 is a noiseless domain so the fitness update can be simplified:

\[
\begin{align*}
p & \leftarrow p + 2 & \text{win} \\
p & \leftarrow p \quad & \text{draw} \\
p & \leftarrow p - 2 & \text{lose} \\
k & \leftarrow k + 2 & \text{if prediction is correct,} \\
k & \leftarrow k - 2 & \text{otherwise}
\end{align*}
\]
Staged learning
- Q-learning learns the step prior to the goal first:

- Abstraction learns the building block first

Affective learning
- Emotion Transformer:
  IF SONAR > 20 mm THEN CURiosity ++
  ELSE CURiosity =

- Emotion Generator:
  IF CURiosity > 30 THEN SPEED <= 25
  ELSE SPEED <= 10

Explore efficiency
- Non-emotional
- Emotional

Knowledge Discovery: Manual Rule Inspection

Michigan-Style
Large rule-population
How to rank rules in order to identify best?
Imprecise rules in noisy data.
Statistical confidence for selecting attributes or rules as being important.

Pittsburgh-Style
Imprecise rules in noisy data.
Statistical confidence for selecting attributes or rules as being important.
Improving Interpretability and Characterization of Heterogeneity

Rule Compaction Algorithm – Testing Accuracy Based
Retain a non-redundant, maximally general subset of the rule population.

Custom Fitness Calculation
Based on a rule’s accuracy, generality, with and strength (age restricted)

Visualization Strategies – Interpreting the Black Box

Visualizing the Rule Population: Interpreting the Black Box – Original UCS

Attributes: 20
Predictive: 4
Non-Predictive: 16
Heritability = 0.4
MAF = 0.2
Sample Size = 1600
Testing Accuracy = 0.70 \( (p = 0.001) \)
Conclusions

Significant Testing Accuracy = 0.7 (p = 0.001)

Our 4 modeled predictive attributes, significantly overrepresented

Our separate epistatic model pairs individually have the largest and most significant CoS's.
Attribute Tracking Scores

Bladder Cancer Analysis: 10 Attributes

Testing Accuracy = 0.60 \((p = 0.001)\)

Individual Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>SpSp</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPD.751</td>
<td>5016</td>
<td>0.001*</td>
</tr>
<tr>
<td>XPD.312</td>
<td>5077</td>
<td>0.001*</td>
</tr>
<tr>
<td>pack.yr</td>
<td>427</td>
<td>0.037*</td>
</tr>
<tr>
<td>XRCC1.194</td>
<td>230</td>
<td>0.179</td>
</tr>
<tr>
<td>age.50</td>
<td>4151</td>
<td>0.721</td>
</tr>
<tr>
<td>male</td>
<td>4046</td>
<td>0.745</td>
</tr>
<tr>
<td>XRCC1.399</td>
<td>3755</td>
<td>0.427</td>
</tr>
<tr>
<td>APE1</td>
<td>3334</td>
<td>0.091</td>
</tr>
<tr>
<td>XRC3</td>
<td>3172</td>
<td>0.096</td>
</tr>
<tr>
<td>XPC.PAT</td>
<td>2975</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Pairs

<table>
<thead>
<tr>
<th>Attribute Pairs</th>
<th>CoS &amp; p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPD.751</td>
<td>3267 0.003*</td>
</tr>
<tr>
<td>XPD.312</td>
<td>2757 0.003*</td>
</tr>
<tr>
<td>XPD.751</td>
<td>pack.yr 230</td>
</tr>
<tr>
<td>XPD.312</td>
<td>pack.yr 2375</td>
</tr>
<tr>
<td>XRCC1.194</td>
<td>age.50 2345</td>
</tr>
<tr>
<td>XRCC1.194</td>
<td>pack.yr 2120</td>
</tr>
<tr>
<td>XPD.312</td>
<td>XRCC1.194 2105</td>
</tr>
</tbody>
</table>

Bladder Cancer Analysis

- Bladder Cancer
  - 4th Men (50,000), 9th Women (16,000)
- 355 Cases, 559 Controls
- 10 Attributes
  - Smoking, Age, Sex
  - 7 DNA Repair Gene SNPs
    - Xeroderma pigmentosum D (XPD)
    - Nucleotide excision repair (NER)
    - XPD 312, XPD 751
- Clinical Variables
  - Age of Diagnosis
  - Survivorship
  - Time to First Recurrence
  - Tumor Stage/Grade

Rule Population: 10 Attributes
Bladder Cancer Analysis: 3 Significant Attributes

XPD 312 & XPD 751 & Pack-Years

Testing Accuracy = 0.70 (p = 0.001)

- Characterizing Heterogeneity with Attribute Tracking
  - Identify Patient Subsets Characterized by Patterns of Association
  - pvclust – Identifies significant, stable clusters via multi-scale bootstrap re-sampling (1000)
  - 7 Significant Clusters (A-G)
  - 2 Particularly Large Clusters (B and D)
  - B – XPD SNPs yield higher tracking scores
  - D – Pack-years yield higher tracking scores

Clinical Variable Analysis:

CASES: Age at Diagnosis

Not Significant

CASES: Age of Recurrence

p = 0.051

CASES: Survivorship

p < 0.05
Bladder Cancer Summary

Smoking Bad

Age of Diagnosis
B – Tends to be diagnosed earlier
D – Tends to be diagnosed later

Marginally Significant Difference in Recurrence
B - Later recurrence
D – Earlier recurrence

Survivorship
B – Survived longer
D – Shorter survival

Tumor Stage/Grade – Too sparse – Not Significant

Replicated Previously Implicated Risk Factors
Potential Novel Pattern of Heterogeneity

LCS Summary

- Rule-based, multifaceted, machine learning algorithms
- Global search and learning through evolution mechanism
- Local search and adaption through reinforcement learning techniques – competition with cooperation
- Multitude of flexible implementations and representations
- Practical applications as now paths through the swamp.

LCSs Applications

- **Reinforcement Learning Problems**
  Find an optimal behavioral policy represented by a compact set of rules.

- **Function Approximation Problems**
  Find an accurate function approximation represented by a partially overlapping set of approximation rules.

- **Classification / Data Mining Problems**
  Find a compact set of rules that classify all problem instances with maximal accuracy.

References