Introducing Rule-Based Machine Learning: A Practical Guide

Ryan J Urbanowicz
University of Pennsylvania
Philadelphia, PA, USA
ryanurb@upenn.edu

Will Browne
Victoria University of Wellington
Wellington, New Zealand
will.browne@vuw.ac.nz

http://www.sigevo.org/gecco-2015/

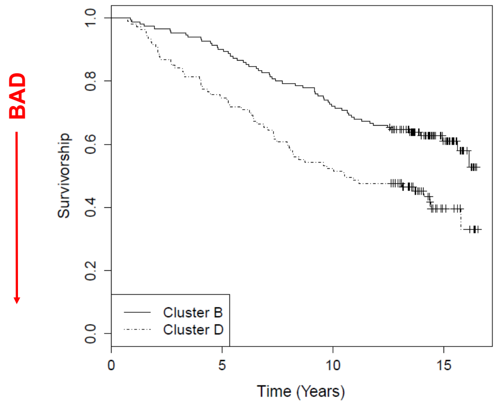
GECCO'15 Companion, July 11–15, 2015, Madrid, Spain.
ACM 978-1-4503-3488-4/15/07.
http://dx.doi.org/10.1145/2739482.2756590

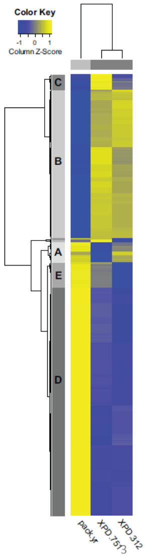
Instructors

- Ryan Urbanowicz is a post-doctoral research associate at the University of Pennsylvania in the Pearlman School of Medicine. He completed a Bachelors and Masters degree in Biological Engineering at Cornell University (2004 & 2005) and a Ph.D in Genetics at Dartmouth College (2012). His research focuses on the methodological development and application of learning classifier systems to complex, heterogeneous problems in bioinformatics, genetics, and epidemiology.

- Will Browne is an Associate Professor at the Victoria University of Wellington. He completed a Bachelors of Mechanical Engineering at the University of Bath, a Masters and EngD from Cardiff, post-doc. Leicester and lecturer in Cybernetics at Reading, UK. His research focuses on applied cognitive systems. Specifically how to use inspiration from natural intelligence to enable computers/ machines/ robots to behave usefully. This includes cognitive robotics, learning classifier systems, and modern heuristics for industrial application.

Bladder Cancer Study: Clinical Variable Analysis-Survivorship





*Images adapted from [1]

Course Agenda

- Introduction (What and Why?)
  - LCS Applications
  - Distinguishing Features of an LCS
  - Historical Perspective
- Driving Mechanisms
  - Discovery
  - Learning
- LCS Algorithm Walk-Through (How?)
  - Rule Population
  - Set Formation
  - Covering
  - Prediction/Action Selection
  - Parameter Updates/Credit Assignment
  - Subsumption
  - Genetic Algorithm
  - Deletion
- Michigan vs. Pittsburgh-style
- Advanced Topics
- Resources

Introduction: What is Rule-Based Machine Learning?

- **Rule Based Machine Learning (RBML)**

- **What types of algorithms fall under this label?**
  - Learning Classifier Systems (LCS)*
    - Michigan-style LCS
    - Pittsburgh-style LCS
  - Association Rule Mining
  - Related Algorithms
    - Artificial Immune Systems

- **Rule-Based** – The solution/model/output is collectively comprised of individual rules typically of the form (IF: THEN).

- **Machine Learning** – "A *subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Explores the construction and study of algorithms that can learn from and make predictions on data." – Wikipedia*

- Keep in mind that machine learning algorithms exist across a continuum.
  - Hybrid Systems
  - Conceptual overlaps in addressing different types of problem domains.

* *LCS algorithms are the focus of this tutorial.*

Introduction: Comparison of RBML Algorithms

- **Learning Classifier Systems (LCS)**
  - Developed primarily for modeling, sequential decision making, classification, and prediction in complex adaptive system .
  - IF:THEN rules link independent variable states to dependent variable states. e.g. $\{V_1, V_2, V_3\} \rightarrow$ Class/Action

- **Association Rule Mining (ARM)**
  - Developed primarily for discovering interesting relations between variables in large datasets.
  - IF:THEN rules link independent variable(s) to some other independent variable e.g. $\{V_1, V_2, V_3\} \rightarrow V_4$

- **Artificial Immune Systems (AIS)**
  - Developed primarily for anomaly detection (i.e. differentiating between self vs. not-self)
  - Multiple `Antibodies' (i.e. detectors) are learned which collectively characterize 'self' or ''not-self' based on an affinity threshold.

- What's in common?
  - In each case, the solution or output is determined piece-wise by a set of `rules' that each cover part of the problem at hand. No single, `model' expression is output that seeks to describe the underlying pattern(s).

- This tutorial will focus on LCS algorithms, and approach them initially from a supervised learning perspective (for simplicity).

Introduction: Why LCS Algorithms? {1 of 3}

- **Adaptive** – Accommodate a changing environment. Relevant parts of solution can evolve/update to accommodate changes in problem space.

- **Model Free** – Limited assumptions about the environment*
  - Can accommodate complex, epistatic, heterogeneous , or distributed underlying patterns.
  - No assumptions about the number of predictive vs. non-predictive attributes (feature selection).

- **Ensemble Learner** (unofficial) – No single model is applied to a given instance to yield a prediction. Instead a set of relevant rules contribute a `vote'.

- **Stochastic Learner** – Non-deterministic learning is advantageous in large-scale or high complexity problems, where deterministic learning becomes intractable.

- **Multi-objective** (Implicitly) – Rules evolved towards accuracy and generality/simplicity.

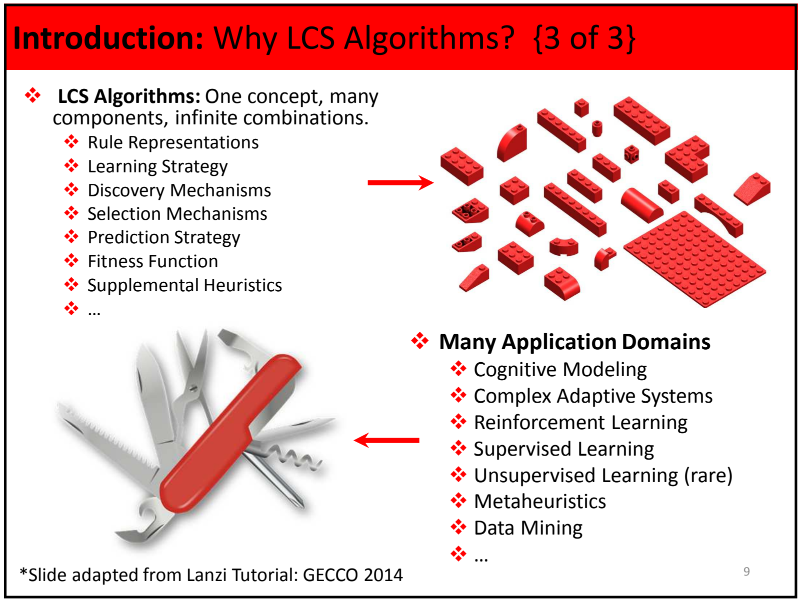
- **Interpretable** (Data Mining/Knowledge Discovery) – Depending on rule representation, individual rules are logical and human readable IF:THEN statements. Strategies have been proposed for global knowledge discovery over the rule population solution [X X].

* *The term `environment' refers to the source of training instances for a problem/task.*

Introduction: Why LCS Algorithms? {2 of 3}

- **Other Advantages**
  - Applicable to single-step or multi-step problems.
  - Representation Flexibility: Can accommodate discrete or continuous-valued endpoints* and attributes (i.e. Dependent or Independent Variables)
  - Can learn in clean or very noisy problem environments.
  - Accommodates missing data (i.e. missing attribute values within training instances).
  - Classifies binary or multi-class discrete endpoints (classification).
  - Can accommodate balanced or imbalanced datasets (classification).

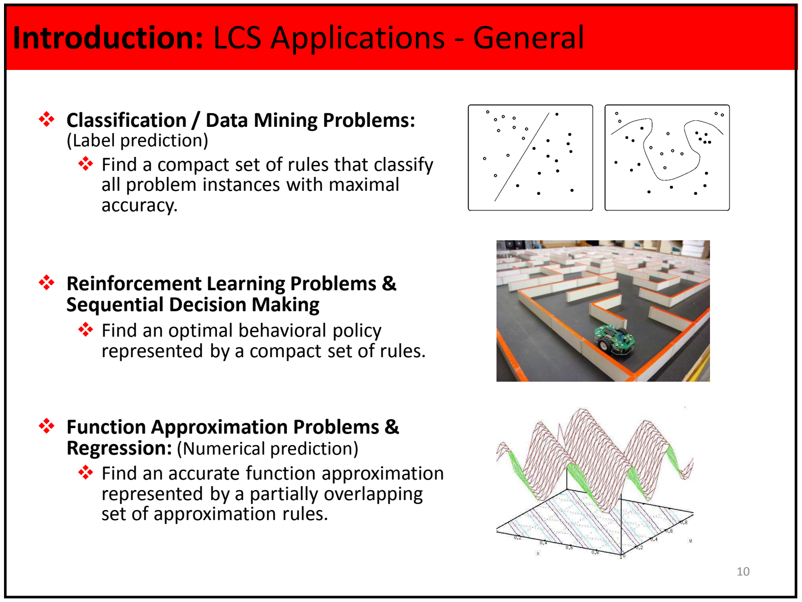
* *We use the term `endpoints' to generally refer to dependent variables .*



Introduction: Why LCS Algorithms? {3 of 3}

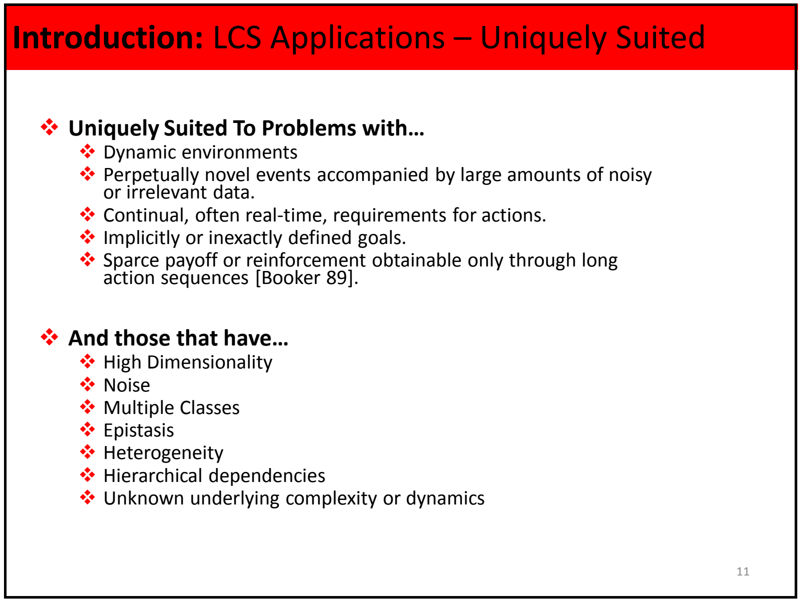
- **LCS Algorithms:** One concept, many components, infinite combinations.
  - Rule Representations
  - Learning Strategy
  - Discovery Mechanisms
  - Selection Mechanisms
  - Prediction Strategy
  - Fitness Function
  - Supplemental Heuristics
  - …
- **Many Application Domains**
  - Cognitive Modeling
  - Complex Adaptive Systems
  - Reinforcement Learning
  - Supervised Learning
  - Unsupervised Learning (rare)
  - Metaheuristics
  - Data Mining
  - …

*Slide adapted from Lanzi Tutorial: GECCO 2014



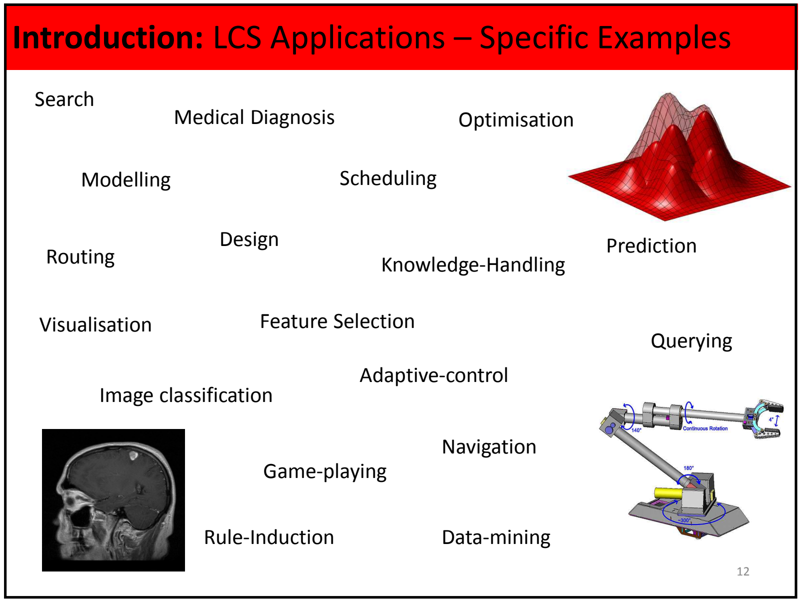
Introduction: LCS Applications - General

- **Classification / Data Mining Problems:** (Label prediction)
  - Find a compact set of rules that classify all problem instances with maximal accuracy.
- **Reinforcement Learning Problems & Sequential Decision Making**
  - Find an optimal behavioral policy represented by a compact set of rules.
- **Function Approximation Problems & Regression:** (Numerical prediction)
  - Find an accurate function approximation represented by a partially overlapping set of approximation rules.



Introduction: LCS Applications – Uniquely Suited

- **Uniquely Suited To Problems with…**
  - Dynamic environments
  - Perpetually novel events accompanied by large amounts of noisy or irrelevant data.
  - Continual, often real-time, requirements for actions.
  - Implicitly or inexactly defined goals.
  - Sparce payoff or reinforcement obtainable only through long action sequences [Booker 89].
- **And those that have…**
  - High Dimensionality
  - Noise
  - Multiple Classes
  - Epistasis
  - Heterogeneity
  - Hierarchical dependencies
  - Unknown underlying complexity or dynamics



Introduction: LCS Applications – Specific Examples

Search

Medical Diagnosis

Optimisation

Modelling

Scheduling

Routing

Design

Knowledge-Handling

Prediction

Visualisation

Feature Selection

Querying

Image classification

Adaptive-control

Navigation

Game-playing

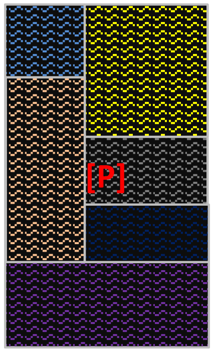
Rule-Induction

Data-mining

Introduction: Distinguishing Features of an LCS

- **Learning Classifier Systems** typically combine:
  - **Global search** of evolutionary computing (e.g. Genetic Algorithm)
  - **Local optimization** of machine learning (supervised or reinforcement)

  THINK: Trial and error meets neo-Darwinian evolution.
- Solution/output is given by a set of IF:THEN rules.
  - Learned patterns are distributed over this set.
  - Output is a distributed and generalized probabilistic prediction model.
  - IF:THEN rules can specify any subset of the attributes available in the environment.
  - IF:THEN rules are only applicable to a subset of possible instances.
  - IF:THEN rules have their own parameters (e.g. accuracy, fitness) that reflect performance on the instances they match.
  - Rules with parameters are termed `classifiers.
- Incremental Learning (Michigan-style LCS)
  - Rules are evaluated and evolved one instance from the environment at a time.
- Online or Offline Learning (Based on nature of environment)



Introduction: Historical Perspective {1 of 5}

1970's → *Genetic algorithms and CS-1 emerge
*Research flourishes, but application success is limited.

1980's

1990's

2000's

2010's

- LCSs are one of the earliest artificial cognitive systems - developed by John Holland (1978). His work at the University of Michigan introduced and popularized the genetic algorithm.
- Holland's Vision: Cognitive System One (CS-1) [2]
  - Fundamental concept of classifier rules and matching.
  - Combining a credit assignment scheme with rule discovery.
  - Function on environment with infrequent payoff/reward.
- The early work was ambitious and broad. This has led to many paths being taken to develop the concept over the following 40 years.
- *CS-1 archetype would later become the basis for `Michigan-style' LCSs.

Introduction: Historical Perspective {2 of 5}

1970's

1980's → *LCS subtypes appear: Michigan-style vs. Pittsburgh-style
*Holland adds reinforcement learning to his system.
*Term `Learning Classifier System' adopted.
*Research follows Holland's vision with limited success.
*Interest in LCS begins to fade.

1990's

2000's

2010's

- Pittsburgh-style algorithms introduced by Smith in Learning Systems One (LS-1) [3]
- Booker suggests niche-acting GA (in [M]) [4].
- Holland introduces bucket brigade credit assignment [5].
- Interest in LCS begins to fade due to inherent algorithm complexity and failure of systems to behave and perform reliably.

Introduction: Historical Perspective {3 of 5}

1970's

1980's

1990's → *REVOLUTION!
*Simplified LCS algorithm architecture with ZCS.
*XCS is born: First reliable and more comprehensible LCS.
*First classification and robotics applications (real-world).

2000's

2010's

- Frey & Slate present an LCS with predictive accuracy fitness rather than payoff-based strength [6].
- Riolo introduces CFCS2, setting the scene for Q-learning like methods and anticipatory LCSs [7].
- Wilson introduces simplified LCS architecture with ZCS, a strength-based system [8].
- **Wilson revolutionizes LCS algorithms with accuracy-based rule fitness in XCS [9].**
- Holmes applies LCS to problems in epidemiology [10].
- Stolzmann introduces anticipatory classifier systems (ACS) [11].

Introduction: Historical Perspective {4 of 5}

- 1970's
- 1980's
- 1990's
- 2000's
- 2010's

- Wilson introduces XCSF for function approximation [12].
- Kovacs explores a number of practical and theoretical LCS questions [13,14].
- Bernado-Mansilla introduce UCS for supervised learning [15].
- Bull explores LCS theory in simple systems [16].
- Bacardit introduces two Pittsburgh-style LCS systems GAssist and BioHEL with emphasis on data mining and improved scalability to larger datasets[17,18].
- Holmes introduces EpiXCS for epidemiological learning. Paired with the first LCS graphical user interface to promote accessibility and ease of use [19].
- Butz introduces first online learning visualization for function approximation [20].
- Lanzi & Loiacono explore computed actions [21].

*LCS algorithm specializing in supervised learning and data mining start appearing.

*LCS scalability becomes a central research theme.

*Increasing interest in epidemiological and bioinformatics.

*Facet-wise theory and applications

Introduction: Historical Perspective {5 of 5}

- 1970's
- 1980's
- 1990's
- 2000's
- 2010's

- Franco & Bacardit explored GPU parallelization of LCS for scalability [22].
- Urbanowicz & Moore introduced statistical and visualization strategies for knowledge discovery in an LCS [23]. Also explored use of `expert knowledge' to efficiently guide GA [24], introduced attribute tracking for explicitly characterizing heterogeneous patterns [25].
- Browne and Iqbal explore new concepts in reusing building blocks (i.e., code fragments) . Solved the 135-bit multiplexer reusing building blocks from simpler multiplexer problems [26].
- Bacardit successfully applied BioHEL to large-scale bioinformatics problems also exploring visualization strategies for knowledge discovery [27].
- Urbanowicz introduced ExSTraCS for supervised learning [28]. Applied ExSTraCS to solve the 135-bit multiplexer directly .

*Increased interest in supervised learning applications persists.

*Emphasis on solution interpretability and knowledge discovery.

*Scalability improving – 135-bit multiplexer solved!

*GPU interest for computational parallelization.

*Broadening research interest from American & European to include Australasian & Asian.

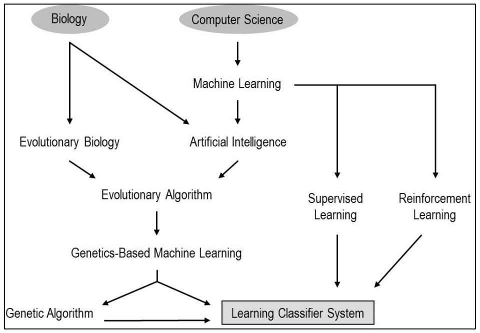
Introduction: Historical Perspective - Summary

- 1970's
- 1980's
- 1990's
- 2000's
- 2010's

- ~40 years of research on LCS has…
  - Clarified understanding.
  - Produced algorithmic descriptions.
  - Determined 'sweet spots' for run parameters.
  - Delivered understandable 'out of the box' code.
  - Demonstrated LCS algorithms to be…
    - Flexible
    - Widely applicable
    - Uniquely functional on particularly complex problems.

Introduction: Naming Convention & Field Tree

- Learning Classifier System (LCS)
  - In retrospect , an odd name.
  - There are many machine learning systems that learn to classify but are not LCS algorithms.
  - E.g. Decision trees
- Also referred to as…
  - Genetics Based Machine Learning (GBML)
  - Adaptive Agents
  - Cognitive Systems
  - Production Systems
  - Classifier System (CS, CFS)



Driving Mechanisms

Two mechanisms are primarily responsible for driving LCS algorithms.

- Discovery
  - Refers to “rule discovery”.
  - Traditionally performed by a genetic algorithm (GA).
  - Can use any directed method to find new rules.
- Learning
  - The improvement of performance in some environment through the acquisition of knowledge resulting from experience in that environment.
  - Learning is constructing or modifying representations of what is being experienced.
  - AKA: Credit Assignment
  - LCSs traditionally utilized reinforcement learning (RL).
  - Many different RL schemes have been applied as well as much simpler supervised learning schemes.

Driving Mechanisms: LCS Rule Discovery {1 of 2}

- Create hypothesised better rules from existing rules & genetic material.
  - Genetic algorithm
    - Original and most common method
    - Well studied
    - Stochastic process
    - The GA used in LCS is most similar to niching GAs
  - 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

Driving Mechanisms: LCS Rule Discovery {2 of 2}

- **When to learn**
  - Too frequent: unsettled [P]
  - Too infrequent: inefficient training
- **What to learn**
  - Most frequent niches or…
  - Underrepresented niches
- **How much to learn**
  - How many good rules to keep (elitism)
  - Size of niche

Driving Mechanisms: Genetic Algorithm (GA)

- Inspired by the neo-Darwinist theory of natural selection, the evolution of rules is modeled after the evolution of organisms using four biological analogies.
  - Genome → Coded Rule (Condition) →
  - Phenotype → Class (Action)
  - Survival of the Fittest → Rule Competition
  - Genetic Operators → Rule Discovery

Example Rules
(Ternary Representation)

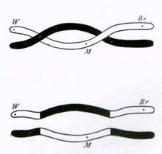
Condition	~	Action
#101#	~	1
#10##	~	0
00#1#	~	0
1#011	~	1

- Elitism (Essential to LCS)
  - LCS preserves the majority of top rules each learning iteration.
  - Rules are only deleted to maintain a maximum rule population size (N).

Driving Mechanisms: GA – Crossover Operator

- Select parent rules

r_1 = 00010001
r_2 = 01110001



- Set crossover point

r_1 = 00|010001
r_2 = 01|110001

- Apply Single Point Crossover

r_1 = 00010001
r_2 = 01110001

c_1 = 00110001
c_2 = 01010001

- Many variations of crossover possible:
  - Two point crossover
  - Multipoint crossover
  - Uniform crossover

Driving Mechanisms: GA – Mutation Operator

- Select parent rule

r_1 = 01110001

- Randomly select bit to mutate

↓
r_1 = 01110001

- Apply mutation

r_1 = 01100001

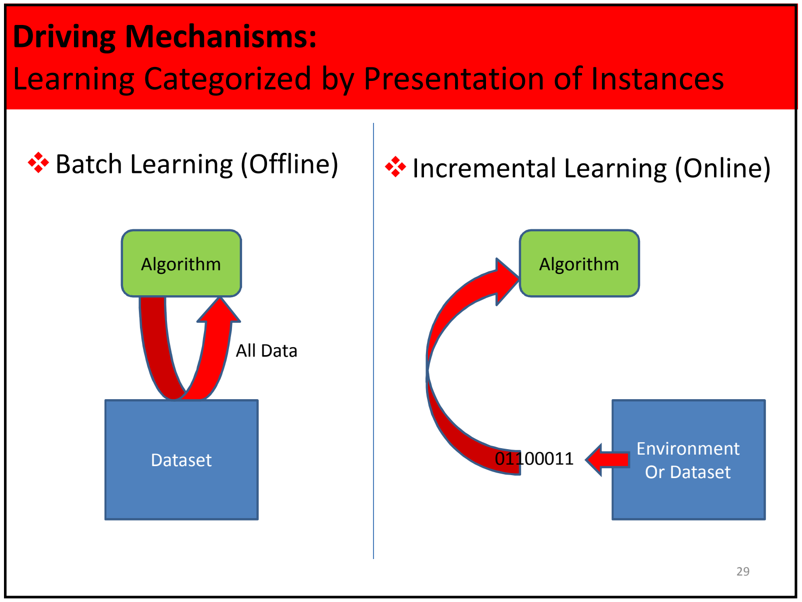
Driving Mechanisms

Two mechanisms are primarily responsible for driving LCS algorithms.

- Discovery
  - Refers to "rule discovery"
  - Traditionally performed by a genetic algorithm (GA)
  - Can use any directed method to find new rules
- Learning
  - The improvement of performance in some environment through the acquisition of knowledge resulting from experience in that environment.
  - Learning is constructing or modifying representations of what is being experienced.
  - AKA: Credit Assignment
  - LCSs traditionally utilized reinforcement learning (RL).
  - Many different RL schemes have been applied as well as much simpler supervised learning (SL) schemes.

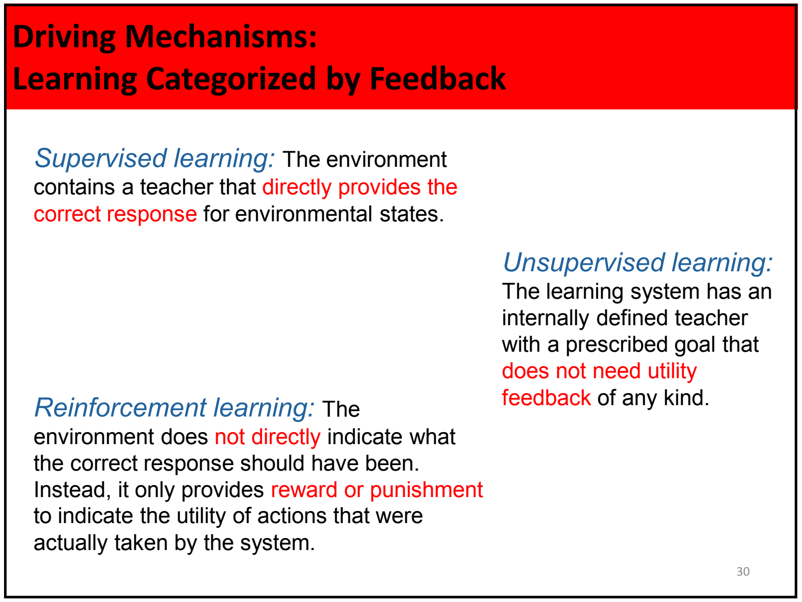
Driving Mechanisms: Learning

- 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.
- Learning strategies can be divided up in a couple ways.
- Categorized by presentation of instances
  - Batch Learning (Offline)
  - Incremental Learning (Online or Offline)
- Categorized by feedback
  - Reinforcement Learning
  - Supervised Learning
  - Unsupervised Learning



Driving Mechanisms:
Learning Categorized by Presentation of Instances

- Batch Learning (Offline)
- Incremental Learning (Online)

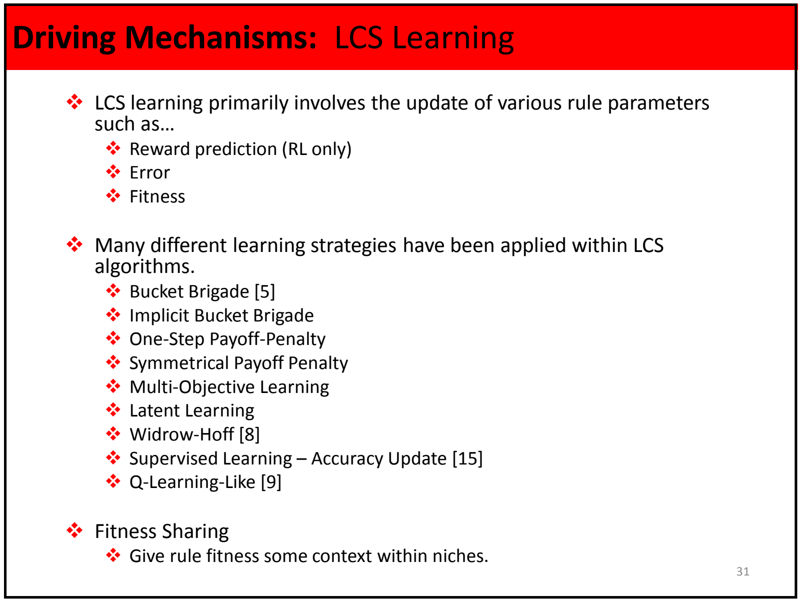


Driving Mechanisms:
Learning Categorized by Feedback

Supervised learning: The environment contains a teacher that directly provides the correct response for environmental states.

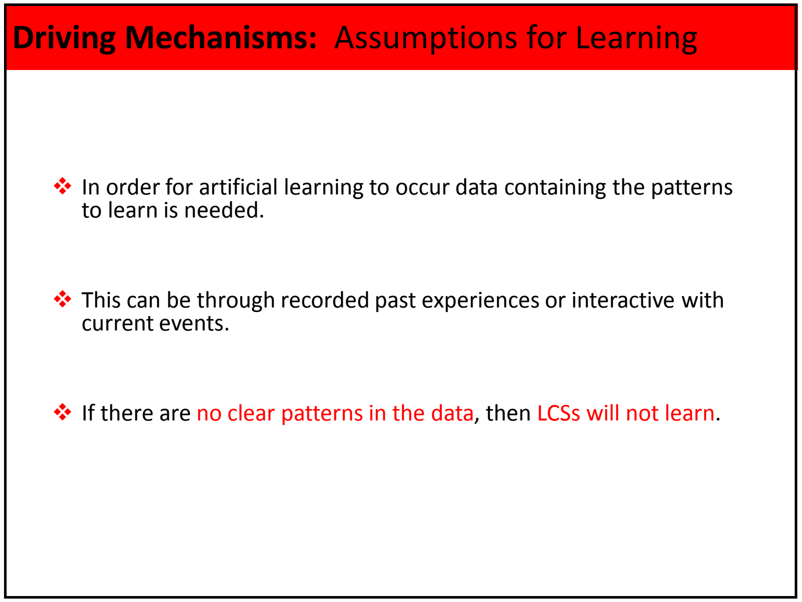
Unsupervised learning: The learning system has an internally defined teacher with a prescribed goal that does not need utility feedback of any kind.

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.



Driving Mechanisms: LCS Learning

- LCS learning primarily involves the update of various rule parameters such as…
  - Reward prediction (RL only)
  - Error
  - Fitness
- Many different learning strategies have been applied within LCS algorithms.
  - Bucket Brigade [5]
  - Implicit Bucket Brigade
  - One-Step Payoff-Penalty
  - Symmetrical Payoff Penalty
  - Multi-Objective Learning
  - Latent Learning
  - Widrow-Hoff [8]
  - Supervised Learning – Accuracy Update [15]
  - Q-Learning-Like [9]
- Fitness Sharing
  - Give rule fitness some context within niches.



Driving Mechanisms: Assumptions for Learning

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

LCS Algorithm Walk-Through

- Demonstrate how a fairly typical modern Michigan-style LCS algorithm…
  - is structured,
  - is trained on a problem environment,
  - makes predictions within that environment
- We use as an example, an LCS architecture most similar to UCS [15], a supervised learning LCS.
- We assume that it is learning to perform a classification/prediction task on a training dataset with discrete-valued attributes, and a binary endpoint.
- We provide discussion and examples beyond the UCS architecture throughout this walk-through to illustrate the diversity of system architectures available.

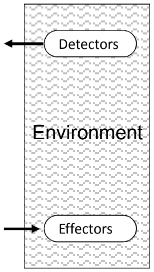
LCS Algorithm Walk-Through: Input {1 of 2}

- Input to the algorithm is often a training dataset.

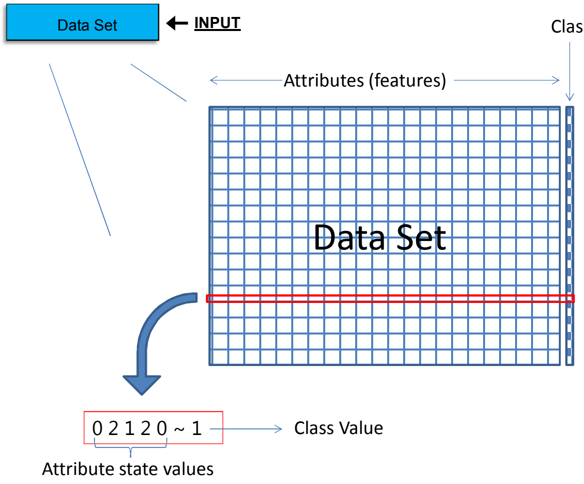
* We will add to this diagram progressively to illustrate components of the LCS algorithm and progress through a typical learning iteration.

LCS Algorithm Walk-Through: Input {2 of 2}

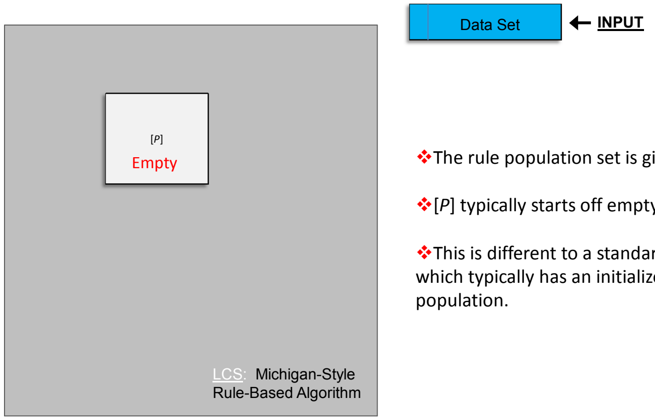
- Detectors
  - Sense the current state of the environment and encode it as a formatted data instance.
  - Grab the next instance from a finite training dataset.
- Effectors
  - Translate action messages into performed actions that modify the state of the environment
- The learning capabilities of LCS rely on and are constrained by the way the agent perceives the environment, e.g., by the detectors the system employs.
- Input data may be binary, integer, real-valued, or some other custom representation, assuming the LCS algorithm has been coded to handle it.



LCS Algorithm Walk-Through: Input Dataset



LCS Algorithm Walk-Through: Rule Population {1 of 2}



- The rule population set is given by [*P*].
- [*P*] typically starts off empty.
- This is different to a standard GA which typically has an initialized population.

LCS Algorithm Walk-Through: Rule Population {2 of 2}

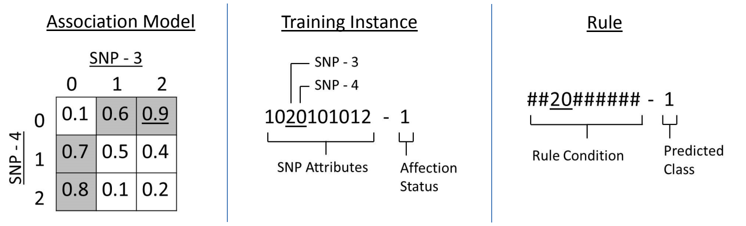
- A finite set of rules [*P*] which collectively explore the 'search space'.
- Every valid rule can be thought of as part of a candidate solution (may or may not be good)
- The space of all candidate solutions is termed the 'search space'.
- The size of the search space is determined by both the encoding of the LCS itself and the problem itself.
- The maximum population size (N) is one of the most critical run parameters.
  - User specified
  - N = 200 to 20000 rules but success depends on dataset dimensions and problem complexity.
  - Too small → Solution may not be found
  - Too large → Run time or memory limits too extreme.

[P]

LCS Algorithm Walk-Through: LCS Rules/Classifiers

Population [*P*]
Classifier$_n$ = Condition : Action :: Parameter(s)

- An analogy:
  - A termite in a mount.
  - A rule on it's own is not a viable solution.
  - Only in collaboration with other rules is the solution space covered.
- Each classifier is comprised of a condition, an action (a.k.a. class, endpoint, or phenotype), and associated parameters (statistics).
- These parameters are updated every learning iteration for relevant rules.



SNP - 4 \ SNP - 3	0	1	2
0	0.1	0.6	0.9
1	0.7	0.5	0.4
2	0.8	0.1	0.2

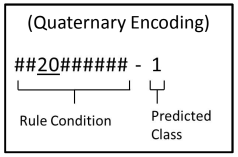
LCS Algorithm Walk-Through: Rule Representation - Ternary

- LCSs can use many different representation schemes.
  - Also referred to as `encodings'
  - Suited to binary input or
  - Suited to real-valued inputs and so forth...
- Ternary Encoding – traditionally most commonly used
  - The ternary alphabet matches binary input
- A attribute in the condition that we don't care about is given the symbol '#' (wild card)
- 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 ' either off on' has action 'on'
- In many binary instances, # acts as an OR function on {0,1}

(Ternary Representation)

Condition	~	Class
# 1 0 1 #	~	1
# 1 0 # #	~	0
0 0 # 1 #	~	0
1 # 0 1 1	~	1

LCS Algorithm Walk-Through: Rule Representation – Other {1 of 4}


(Quaternary Encoding)

##2<u>0</u>###### - 1

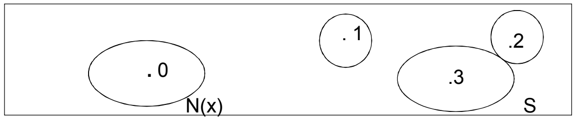
Rule Condition

Predicted Class

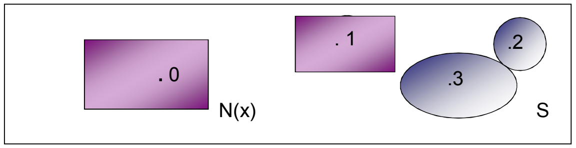
- Quaternary Encoding [29]
  - 3 possible attribute states {0,1,2} plus '#'.
  - For a specific application in genetics.
- Real-valued interval (XCSR [30])
  - Interval is encoded with two variables: center and spread
  - i.e. [center,spread] → [center-spread, center+spread]
  - i.e. [0.125,0.023] → [0.097, 0.222]
- Real-valued interval (UBR [31])
  - Interval is encoded with two variables: lower and upper bound
  - i.e. [lower, upper]
  - i.e. [0.097, 0.222]
- Messy Encoding (Gassist, BIOHel, ExSTraCS [17,18,28])
  - Attribute-List Knowledge Representation (ALKR) [33]
  - 11##0:1 shorten to 110:1 with reference encoding
  - Improves transparency, reduces memory and speeds processing

---

LCS Algorithm Walk-Through: Rule Representation – Other {2 of 4}



- 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 ≤ *x* ≤ 10, which works fine in this case...

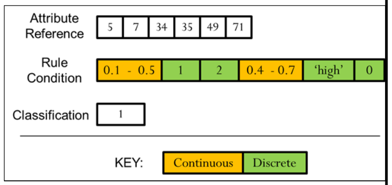


- We form Hypercubes with the number of dimensions = the number of conditions.
- Approximates actual niches, so Classes 2 & 3 difficult to separate with this encoding, so use Hyperellipsoids

---

LCS Algorithm Walk-Through: Rule Representation – Other {3 of 4}

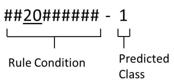
- Mixed Discrete-Continuous ALKR [28]
  - Useful for big and data with multiple attribute types
    - Discrete (Binary, Integer, String)
    - Continuous (Real-Valued)
  - Similar to ALKR (Attribute List Knowledge Representation): [Bacardit et al. 09]
  - Intervals used for continuous attributes and direct encoding used for discrete.



Attribute Reference	5	7	34	35	49	71
Rule Condition	0.1 - 0.5	1	2	0.4 - 0.7	'high'	0
Classification	1					

KEY: Continuous Discrete

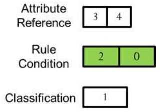
<u>Ternary</u>


##2<u>0</u>###### - 1

Rule Condition

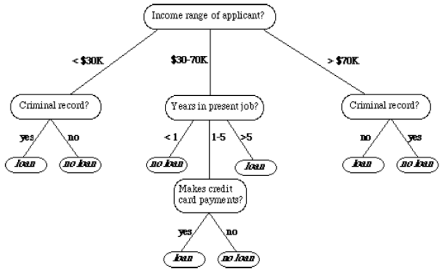
Predicted Class

<u>Mixed</u>



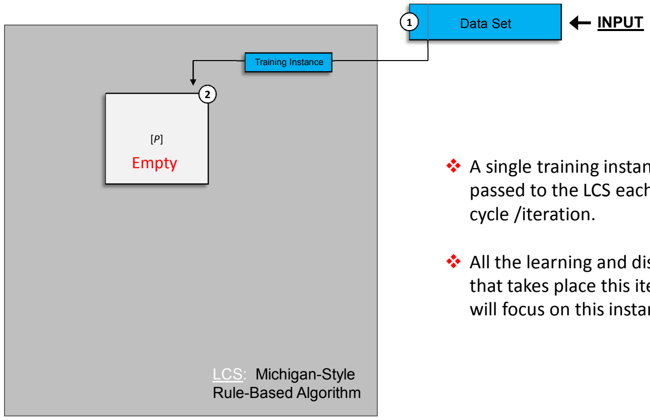
Attribute Reference	3	4
Rule Condition	2	0
Classification	1	

LCS Algorithm Walk-Through: Rule Representation – Other {4 of 4}



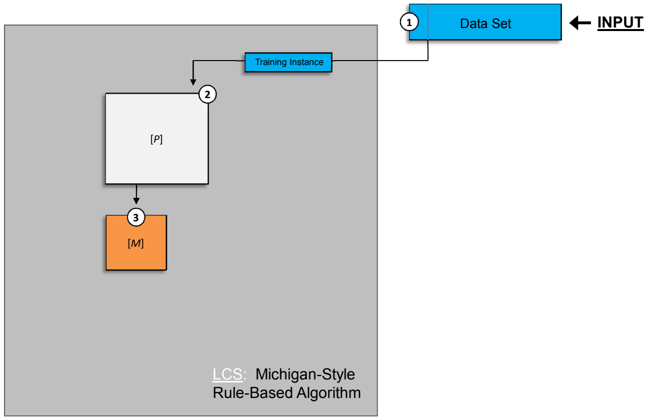
- Decision trees [32]
- Code Fragments [26]
- Artificial neural network
- Fuzzy logic/sets
- Horn clauses and logic
- S-expressions, GP-like trees and code fragments.
- NOTE – Alternative action encodings also utilized
  - Computed actions – replaces action value with a function [21]

LCS Algorithm Walk-Through: Get Training Instance



- A single training instance is passed to the LCS each learning cycle /iteration.
- All the learning and discovery that takes place this iteration will focus on this instance.

LCS Algorithm Walk-Through: Form Match Set [M]

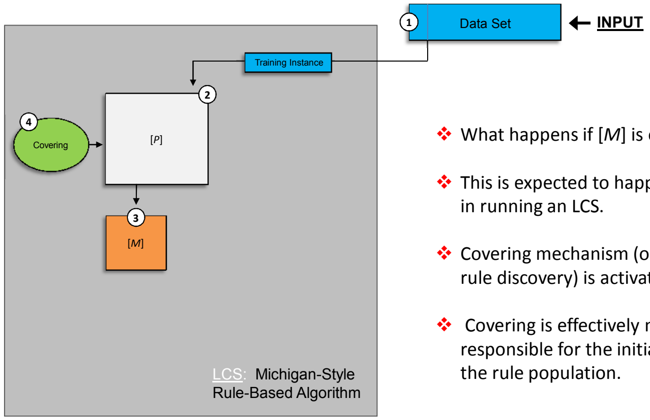


LCS Algorithm Walk-Through: Matching

[*M*]

- How do we form a match set?
  - Find any rules in [*P*] that match the current instance.
  - A rule matches an instance if…
    - All attribute states specified in the rule equal or include the complementary attribute state in the instance.
    - A `#' (wild card) will match any state value in the instance.
  - All matching rules are placed in [*M*].
- What constitutes a match?
  - **Given:** An instance with 4 binary attributes states `1101' and class 1.
  - **Given:** Rule$_a$ = 1##0 ~ 1
  - The first attribute matches because the '1' specified by Rule$_a$ equals the '1' for the corresponding attribute state in the instance.
  - The second attributes because the '#' in Rule$_a$ matches state value for that attribute.
- Note: Matching strategies are adjusted for different data/rule encodings.

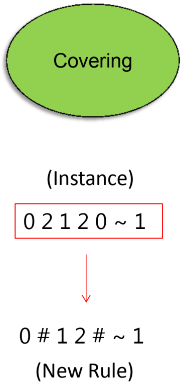
LCS Algorithm Walk-Through: Covering {1 of 2}



- What happens if [*M*] is empty?
- This is expected to happen early on in running an LCS.
- Covering mechanism (one form of rule discovery) is activated.
- Covering is effectively most responsible for the initialization of the rule population.

LCS Algorithm Walk-Through: Covering {2 of 2}

- Covering initializes a rule by generalizing an instance.
  - **Condition**: Generalization of instance attribute states.
  - **Class**:
    - If supervised learning: Assigned correct class
    - If reinforcement learning: Assigned random class/action
- Covering adds #'s to a new rule with probability of generalization ($P_{\#}$) of 0.33 - 0.5 (common settings).
- New rule is assigned initial rule parameter values.
- NOTE: Covering will only add rules to the population that match at least one data instance.
  - This avoids searching irrelevant parts of the search space.



(Instance)

0 2 1 2 0 ~ 1

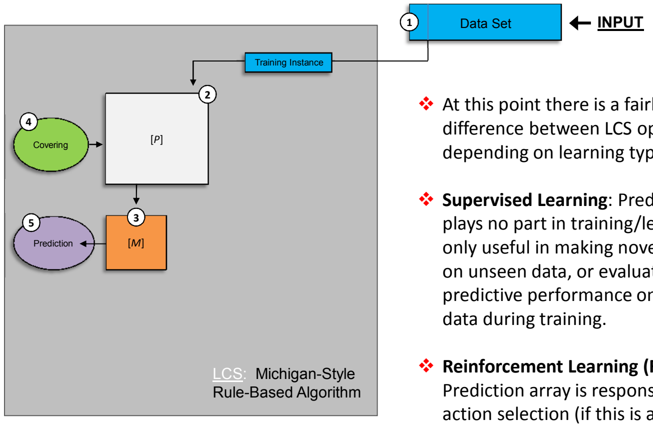
0 # 1 2 # ~ 1

(New Rule)

LCS Algorithm Walk-Through: Special Cases for Matching and Covering

- Matching:
  - **Continuous-valued attributes**: Specified attribute interval in rule must include instance value for attribute. E.g. [0.2, 0.5] includes 0.34.
  - Alternate strategy-
    - Partial match of rule is acceptable (e.g. 3/4 states). Might be useful in high dimensional problem spaces.
- Covering:
  - **For supervised learning** – also activated if no rules are found for [*C*]
  - Alternate activation strategies-
    - Having an insufficient number of matching classifiers for:
      - Given class (Good for best action mapping)
      - All possible classes (Good for complete action mapping and reinforcement learning)
  - Alternate rule generation-
    - Rule specificity limit covering [28]:
      - Removes need for $P_{\#}$, useful/critical for problems with many attributes or high dimensionality.
      - Picks some number of attributes from the instance to specify up to a dataset-dependent maximum.

LCS Algorithm Walk-Through:Prediction Array {1 of 2}



- At this point there is a fairly big difference between LCS operation depending on learning type.
- **Supervised Learning**: Prediction array plays no part in training/learning. It is only useful in making novel predictions on unseen data, or evaluating predictive performance on training data during training.
- **Reinforcement Learning (RL):** Prediction array is responsible for action selection (if this is an exploit iteration).

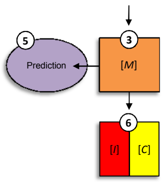
LCS Algorithm Walk-Through:Prediction Array {2 of 2}

- Rules in [*M*] advocate for different classes!
- Want to predict a class (known as action selection in RL).
- In SL, prediction array just makes prediction.
- In RL, prediction array choses predicted action during exploit phase. A random action is chosen for explore phases. This action is sent out into the environment. All rules in [*M*] with this chosen action forms the action set [*A*].
- Consider the fitness (F) of the rules in an SL example.

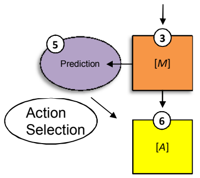
Rule$_a$ 1##101 ~ 1 *F* = 0.8,
Rule$_b$ 1#0##1 ~ 0 *F* = 0.3,
Rule$_c$ 1##1#1 ~ 0 *F* = 0.4, …

- Class/Action can be selected:
  - Deterministically – Class of classifier with best F in [*M*].
  - Probabilistically – Class with best average F across rules in [*M*], i.e. Classifiers vote for the best class.

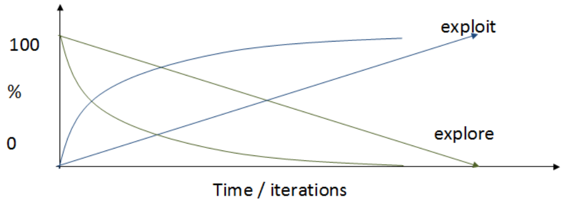
Supervised Learning (SL)



Reinforcement Learning (RL)

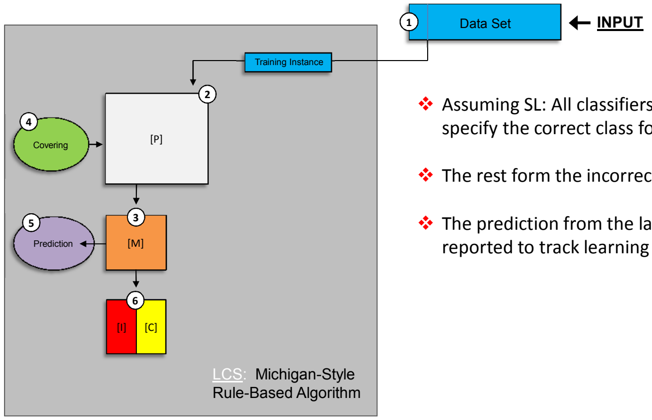


LCS Algorithm Walk-Through: RL - Explore vs. Exploit



- One of the biggest problems in evolutionary computation…
  - When to exploit the knowledge that is being learned?
  - When to explore to learn new knowledge?
- LCS algorithms commonly alternate between explore and exploit for each iteration (incoming data instance).
- In SL based LCS, there is no need to separate explore and exploit iterations. Every iteration: a prediction array is formed, the [C] is formed (since we know the correct class of the instance), and the GA can discover new rules.

LCS Algorithm Walk-Through: Form Correct Set [C]



- Assuming SL: All classifiers in [M] that specify the correct class form [C].
- The rest form the incorrect set [I].
- The prediction from the last set can be reported to track learning progress.

LCS Algorithm Walk-Through: Example [M] and [C]

Data Instance

0 2 1 2 0 ~ 1

Rules

2 # 1 # # ~ 1
2 1 # 0 ~ 1
1 2 # ~ 0

[C]

[I]

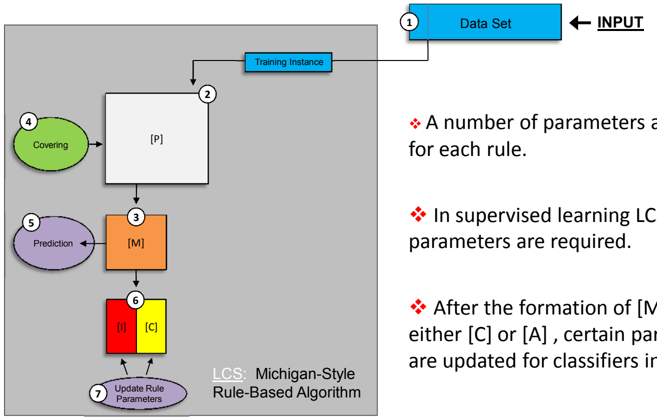
Sample Instance from Training Set

Match Set

Correct Set

0#12# ~ 0	#1211 ~ 0	1#22# ~ 1	2##2# ~ 0
2#1## ~ 1	10102 ~ 0	###20 ~ 0	221## ~ 1
###02 ~ 0	22##2 ~ 1	#0#2# ~ 1	##100~ 1
0#1## ~ 1	####0 ~ 0	#21#0 ~ 1	#122# ~ 0
#2##1 ~ 1	#101# ~ 1	22#1# ~ 0	01### ~ 1
##### ~ 0	2#2## ~ 1	#1### ~ 0	##2## ~ 0
02##0~ 1	010## ~ 0	####2 ~ 1	##00# ~ 1
##12# ~ 0	##2#0 ~ 0	##12# ~ 1	0###0 ~ 0

LCS Algorithm Walk-Through:
Update Rule Parameters / Credit Assignment {1 of 2}



- A number of parameters are stored for each rule.
- In supervised learning LCS, fewer parameters are required.
- After the formation of [M] and either [C] or [A] , certain parameters are updated for classifiers in [M].

LCS Algorithm Walk-Through:
Update Rule Parameters / Credit Assignment {2 of 2}

- An action/class has been chosen and passed to the environment.

Update Rule Parameters

- Supervised Learning:
  - Parameter Updates:
    - Rules in [C] get boost in accuracy.
    - Rules in [M] that didn't make it to [C] get decreased in accuracy.
- Reinforcement Learning:
  - A reward may be returned from the environment
  - RL parameters are updated for rules in [M] and/or [A]

LCS Algorithm Walk-Through:
Update Rule Parameters / Credit Assignment for SL

- Experience is increased in all rules in [M]
- Accuracy is calculated, e.g. UCS

$$acc = \frac{\text{number of correct classifications}}{\text{experience}}$$

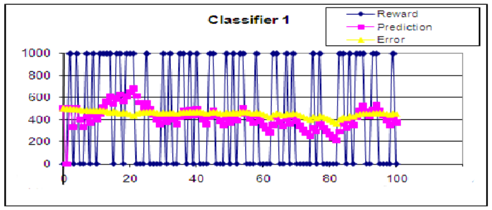
- Fitness is computed as a function of accuracy:

$$F = (acc)^{v}$$

- v used to separate similar fitness classifiers
  - Often set to 10 (in problems assuming without noise)
  - Pressure to emphasize importance of accuracy

LCS Algorithm Walk-Through:
Credit Assignment for RL

- Recency weighted update for prediction.



- Widrow-Hoff update: learning rate β
  value_{new} = value + β x (signal - value)
- Filters the 'noise' in the reward signal
  β = 1 the new value is signal, β = 0 then old value kept

LCS Algorithm Walk-Through:
Credit Assignment for RL + Fitness Sharing

- Classifier considered accurate if:
  - Error < tolerance, otherwise scaled.
- Accuracy relative to action set
- Fitness based on relative accuracy, e.g. XCS

$$p \leftarrow p + \beta(R - p),$$

$$\varepsilon \leftarrow \varepsilon + \beta(|R - p| - \varepsilon),$$

$$\kappa = \begin{cases} 1 & \text{if } \varepsilon < \varepsilon_0 \\ \alpha(\varepsilon/\varepsilon_0)^{-\nu} & \text{otherwise} \end{cases},$$

$$\kappa' = \frac{\kappa}{\sum_{x \in [A]} \kappa_x},$$

$$F \leftarrow F + \beta(\kappa' - F)$$

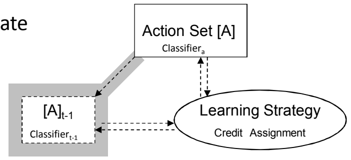
LCS Algorithm Walk-Through:
Credit Assignment for RL + Deferred Reward

- Prediction *p* is updated as follows:

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

where γ is the discount factor
r is reward, β is learning rate
s is state, *a* is action

- Compare this with Q-learning



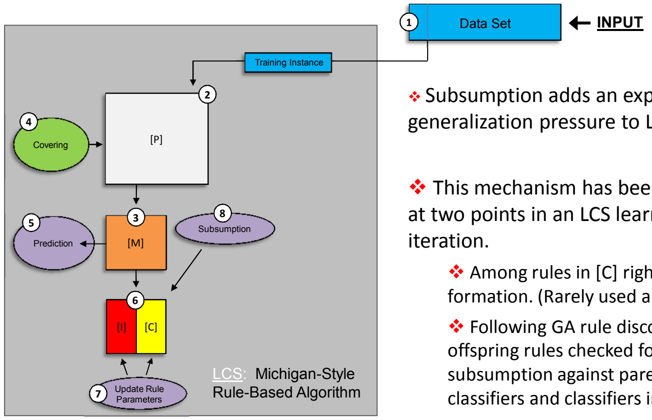
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q^*(s',a') - Q(s,a)]$

where α is learning rate

LCS Algorithm Walk-Through:
Why not Strength-based Fitness?

- Different niches of the environment usually have different payoff levels - Phenotypic niche
- In fitness sharing classifier's strength no longer correctly predicts payoff - Fitness sharing prevents takeover
- Fitness sharing does not prevent more renumerative niches gaining more classifiers - Niche rule discovery helps
- Rule discovery cannot distinguish an accurate classifier with moderate payoff from an overly general classifier having the same payoff on average – Over-generals proliferate
- No reason for accurate generalizations to evolve
- Unnecessarily specific rules survive

LCS Algorithm Walk-Through: Subsumption {1 of 2}



LCS: Michigan-Style Rule-Based Algorithm

- Subsumption adds an explicit rule generalization pressure to LCS.
- This mechanism has been applied at two points in an LCS learning iteration.
  - Among rules in [C] right after its formation. (Rarely used anymore)
  - Following GA rule discovery offspring rules checked for subsumption against parent classifiers and classifiers in [C].

LCS Algorithm Walk-Through: Subsumption {2 of 2}

- In sparse or noisy environments over-specific rules can take over population.

  Want → 10011###1~1

  But got → 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%)
  - **If** rule A is completely accurate ($\varepsilon < \varepsilon_0$) **Then** can delete rule B from the population without loss of performance
- Subsumption mechanisms:
  - GA subsumption
  - Action set [A] subsumption

- Subsumption = General rule (A) absorbs a more specific one (B)
  - Increases rule numerosity

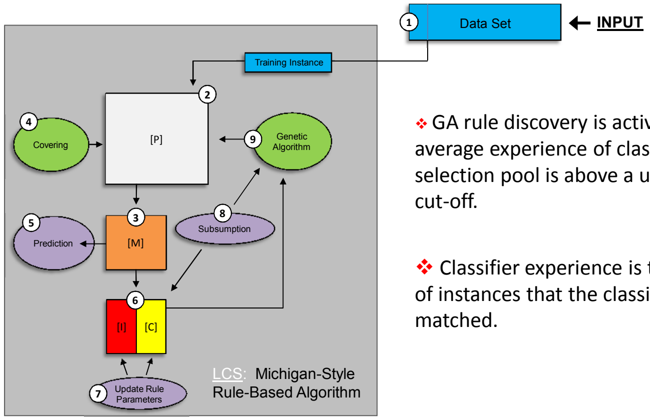
LCS Algorithm Walk-Through: Numerosity {1 of 2}

- 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

LCS Algorithm Walk-Through: Numerosity {2 of 2}

- Numerosity (n) affects action selection and update procedures:
- The fitness sums take numerosity into account:
- Terminology:
  - Macroclassifiers: all unique classifiers n ≥ 1
  - Microclassifiers: all individual classifiers (n copies of macroclassifiers)
- Ratio of macroclassifiers to microclassifiers often used as a measure of training progress.
- Numerosity is also often applied as a `best-available' strategy to ranking rules for manual rule inspection (i.e. knowledge discovery).

LCS Algorithm Walk-Through: Genetic Algorithm



- GA rule discovery is activated if average experience of classifiers in selection pool is above a user defined cut-off.
- Classifier experience is the number of instances that the classifier has matched.

LCS Algorithm Walk-Through: Genetic Algorithm – Other Considerations

Genetic Algorithm

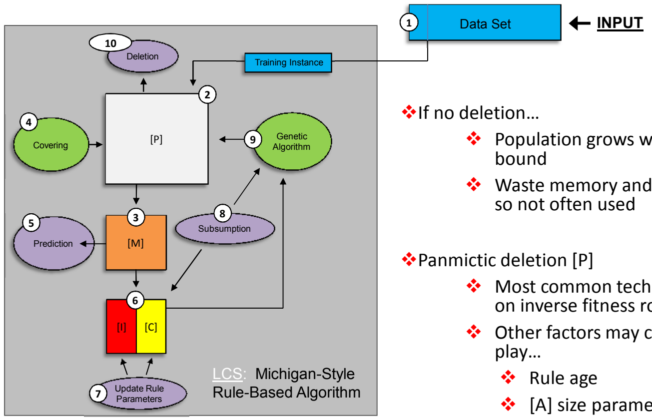
- Parent Selection (typically 2 parents selected)

- Selection Pool:
  - Panmictic – Parents selected from [P] [34]
  - Niche – Parents selected from [M], [4]
  - Refined Niche – Parents selected from [C] or [A], [9]

- Niche GA (Closest to LCS GA)
  - Niching GAs developed for multi modal problems
  - Maintain population diversity to promote identification of multiple peaks
  - Fitness sharing – pressure to deter aggregation of too many 'similar' rules

- Selection Strategy:
  - Deterministic – Pick rules with best fitness from pool.
  - Random – rarely used
  - Probabilistic –
    - Roulette Wheel
    - Tournament Selection

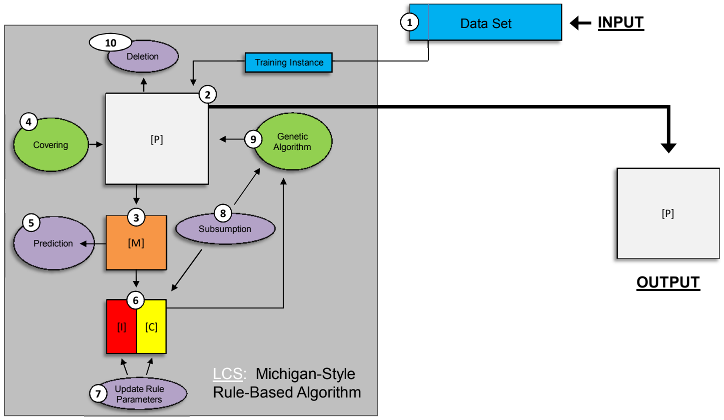
LCS Algorithm Walk-Through: Deletion



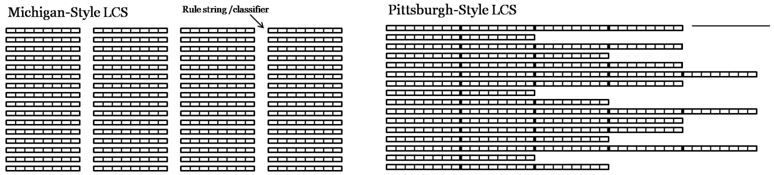
- If no deletion…
  - Population grows without bound
  - Waste memory and takes time so not often used

- Panmictic deletion [P]
  - Most common technique based on inverse fitness roulette wheel
  - Other factors may come into play…
    - Rule age
    - [A] size parameter

LCS Algorithm Walk-Through: Michigan LCS Algorithm



Michigan vs. Pittsburgh-Style LCSs: Major Variations



- Entire population is the solution
- Learns iteratively
- GA operates between individual rules

- Single rule-set is the solution
- Learns batch-wise
- GA operates between rule-sets

Michigan vs. Pittsburgh-Style LCSs: Implementations

- Michigan Style LCS
  - ZCS (Strength Based)
  - XCS (Accuracy Based – Most popular)
  - UCS (Supervised Learning)
  - ACS (Anticipatory)
  - ExSTraCS (Extended Supervised Tracking and Learning)
- Pittsburgh Style LCS
  - GALE (Spatial Rule Population)
  - GAssist (Data mining – Pitt Style Archetype)
  - BIOHEL (Focused on Biological Problems and Scalability)
- Other Hybrid Styles also exist!

LCS Disadvantages

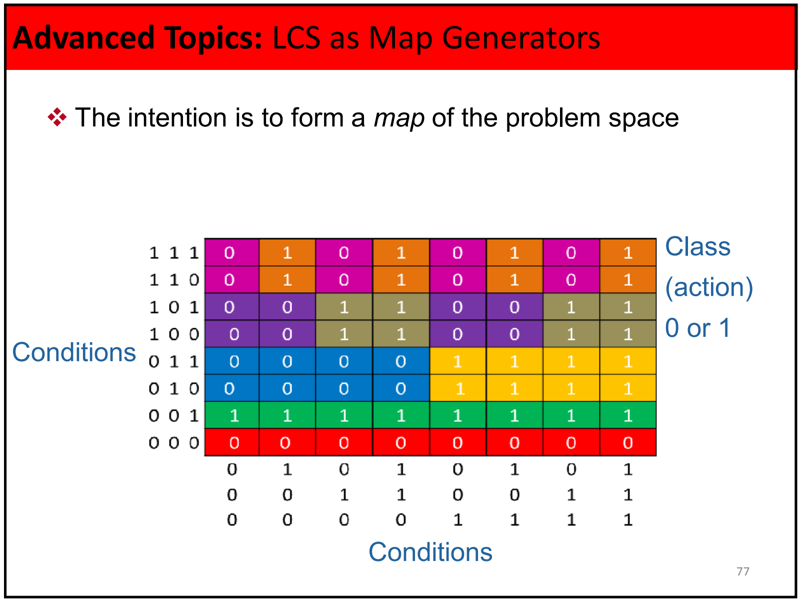
- Not widely known.
- Relatively limited software accessibility.
- Rule population interpretation and knowledge extraction can be challenging.
- Can suffer from overfitting, despite explicit and implicit pressures to generalize rules.
- Relatively little theoretical work or convergence proofs.
- Many run parameters to consider/optimize.

Advanced Topics: Learning Parameters {1 of 2}

Symbol	Meaning
N	Maximum size of the population (In micro-classifiers, N is the number sum of the classifier numerosities)
β	Learning rate for p, ε, f, and as
α, ε_0, ν	Used in calculating the fitness of a classifier
γ	Discount factor used (in multi-step problems) in updating classifier predictions
θ_{GA}	GA threshold The GA is applied when the average time since the last GA in the lest is greater than the threshold.
χ	Probability of applying crossover in GA
μ	Probability of mutating an allele in the offspring
θ_{del}	Deletion threshold If the experience of a classifier is greater than θ_{del}, its fitness may be considered in its probability of deletion
δ	Mean fitness in [P] below which the fitness of a classifier may be considered in its probability of deletion
θ_{sub}	Subsumption threshold – the experience of a classifier must be greater than θ_{sub} in order to be able to subsume another classifier

Advanced Topics: Learning Parameters {2 of 2}

Symbol	Meaning
$P_{\#}$	Probability of using a # in one attribute in C when covering
p_I ε_I F_I	Initial Values in new classifiers
p_{explr}	Probability during action selection of choosing the action uniform randomly
θ_{mna}	Minimal number of actions that must be present in a match set [M] or else covering will occur
doGASumpution is a Boolean parameter that specifies if offspring are to be tested for possible logical subsumption by parents.	
doActionSetSubsumption is a boolean parameter that specifies if action sets are to be tested for subsuming classifiers.	



Advanced Topics: LCS as Map Generators

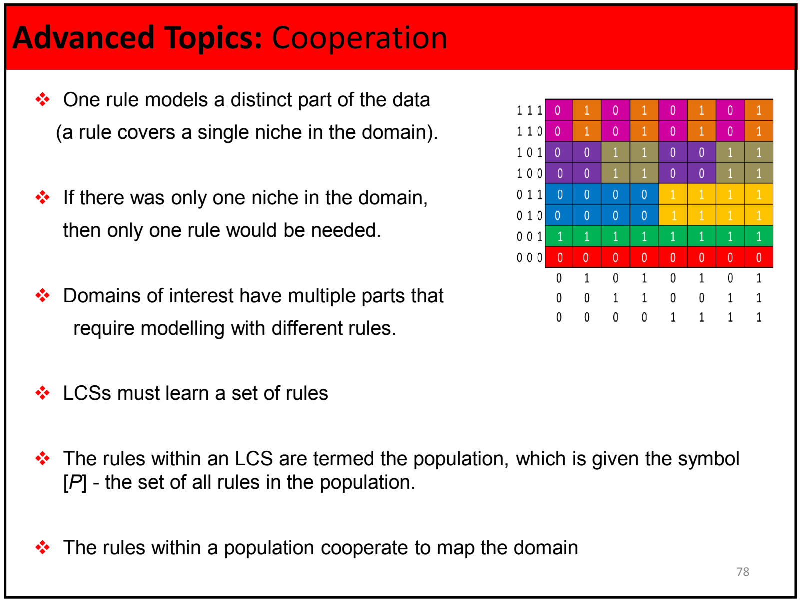
- The intention is to form a *map* of the problem space

Conditions

1 1 1	0	1	0	1	0	1	0	1
1 1 0	0	1	0	1	0	1	0	1
1 0 1	0	0	1	1	0	0	1	1
1 0 0	0	0	1	1	0	0	1	1
0 1 1	0	0	0	0	1	1	1	1
0 1 0	0	0	0	0	1	1	1	1
0 0 1	1	1	1	1	1	1	1	1
0 0 0	0	0	0	0	0	0	0	0
	0	1	0	1	0	1	0	1
	0	0	1	1	0	0	1	1
	0	0	0	0	1	1	1	1

Class
(action)
0 or 1

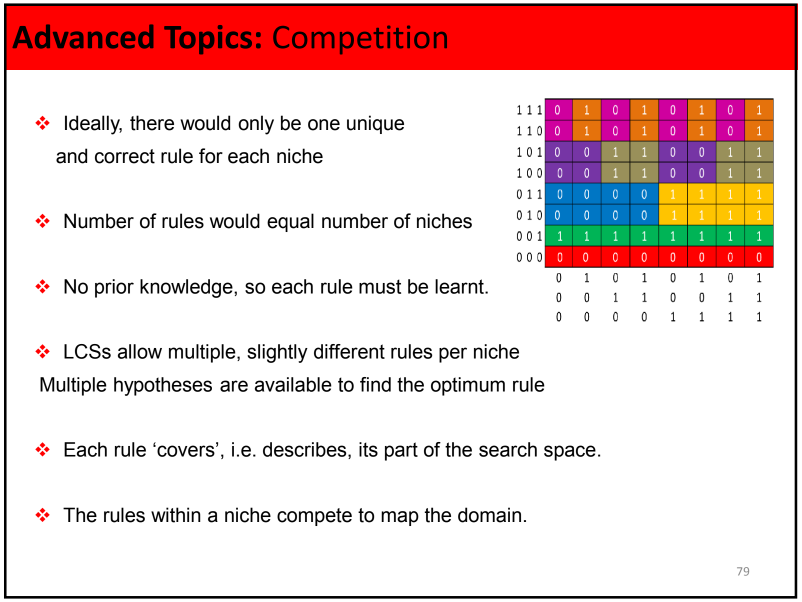
Conditions



Advanced Topics: 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

1 1 1	0	1	0	1	0	1	0	1
1 1 0	0	1	0	1	0	1	0	1
1 0 1	0	0	1	1	0	0	1	1
1 0 0	0	0	1	1	0	0	1	1
0 1 1	0	0	0	0	1	1	1	1
0 1 0	0	0	0	0	1	1	1	1
0 0 1	1	1	1	1	1	1	1	1
0 0 0	0	0	0	0	0	0	0	0
	0	1	0	1	0	1	0	1
	0	0	1	1	0	0	1	1
	0	0	0	0	1	1	1	1



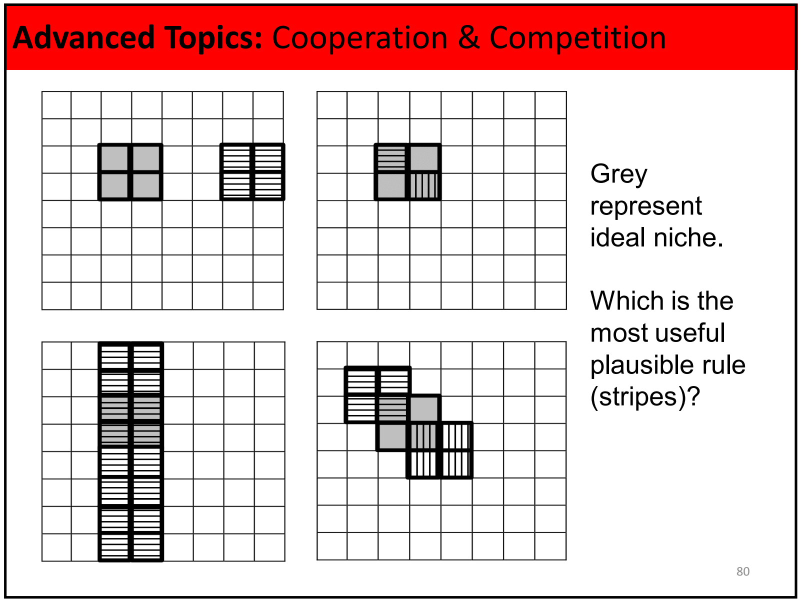
Advanced Topics: Competition

- 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.

1 1 1	0	1	0	1	0	1	0	1
1 1 0	0	1	0	1	0	1	0	1
1 0 1	0	0	1	1	0	0	1	1
1 0 0	0	0	1	1	0	0	1	1
0 1 1	0	0	0	0	1	1	1	1
0 1 0	0	0	0	0	1	1	1	1
0 0 1	1	1	1	1	1	1	1	1
0 0 0	0	0	0	0	0	0	0	0
	0	1	0	1	0	1	0	1
	0	0	1	1	0	0	1	1
	0	0	0	0	1	1	1	1



Advanced Topics: Cooperation & Competition

Grey represent ideal niche.

Which is the most useful plausible rule (stripes)?

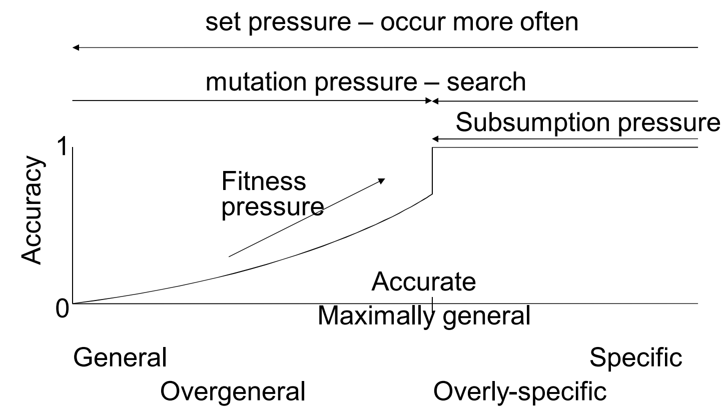
Advanced Topics: Over-generals

- Over-generals are undesired, inaccurate rules that typically match many instances.
- 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 ε_0 is set too high.

Advanced Topics: LCS Pressures



*Adapted from Butz '10

Advanced Topics: 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)

Advanced Topics: 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

Advanced Topics: Mutation Pressure

- Genotypically change the specificity-generality balance

- Mutation can

Randomise:	Generalise:	Specialise:
0 ← 1 or #	0 ← #	0 ← 0
1 ← 0 or #	1 ← #	1 ← 1
# ← 0 or 1	# ← #	# ← 0 or 1

Advanced Topics: Complete vs. Best Action Mapping

- Should LCS discover:
  - The most optimum action in a niche or
  - The predicted payoff for all actions in a niche

X x A => 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

Advanced Topics: LCS Scalability

- What is scalability?
  - Maintaining algorithm tractability as problem scale increases.
  - Problem scale increases can include…
    - Higher pattern dimensionality
    - Larger-scale datasets with
      - Increased number of potentially predictive attributes.
      - Increased number of training instances.

- Strategies for improving LCS scalability.
  - More efficient rule representations [18,28] (Pittsburgh and Michigan)
  - Windowing [36] (Pittsburgh)
  - Computational Parallelization (GPGPUs) [22]
  - Ensemble learning with available attributes partitioned into subsets [27]
  - Expert knowledge guided GA [25]
  - Rule Specificity Limit [28]

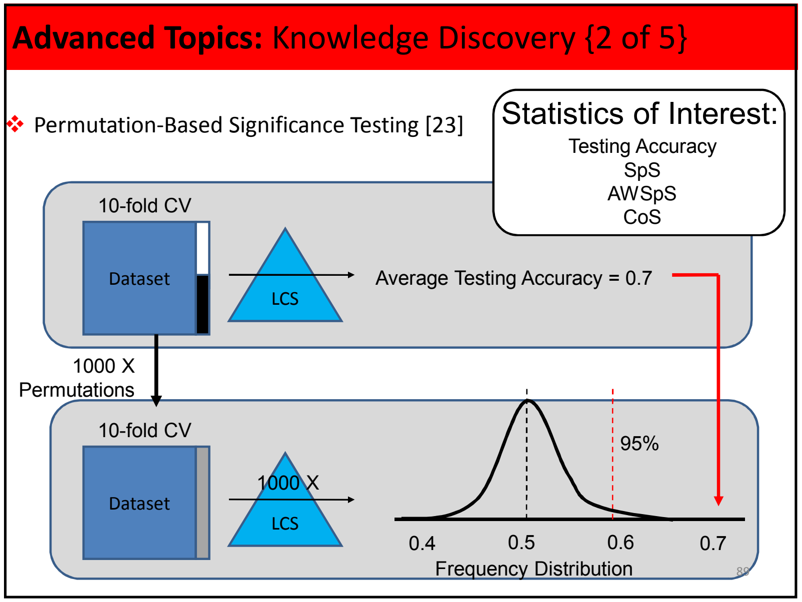
Advanced Topics: Knowledge Discovery {1 of 5}

- Description of global summary statistics for [P] (SpS, AWSpS) [23]

Attribute	X1	X2	X3	X4	Class	Numerosity	Accuracy
R1	X	#	#	X	0	5	0.73
R2	#	X	#	X	1	1	0.51
R3	X	X	#	X	0	2	0.88
R4	#	X	X	#	1	1	0.62
SpS	7	4	1	8			
AWSpS	5.41	2.89	0.62	5.92			

SpS (X1) = 5 + 2 = <u>7</u>

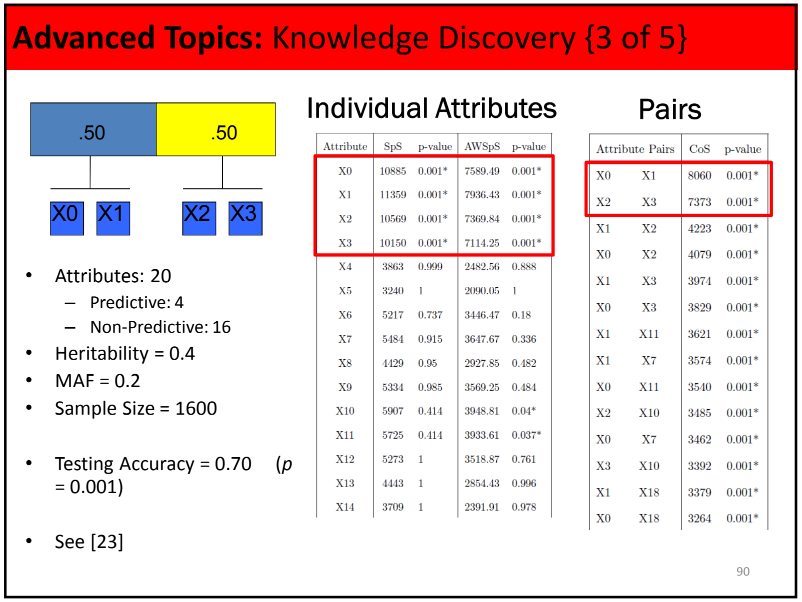
AWSpS (X1) = (0.73) * 5 + (0.88) * 2 = <u>5.41</u>



Advanced Topics: Knowledge Discovery {2 of 5}

- Permutation-Based Significance Testing [23]

Statistics of Interest:
Testing Accuracy
SpS
AWSpS
CoS



Advanced Topics: Knowledge Discovery {3 of 5}

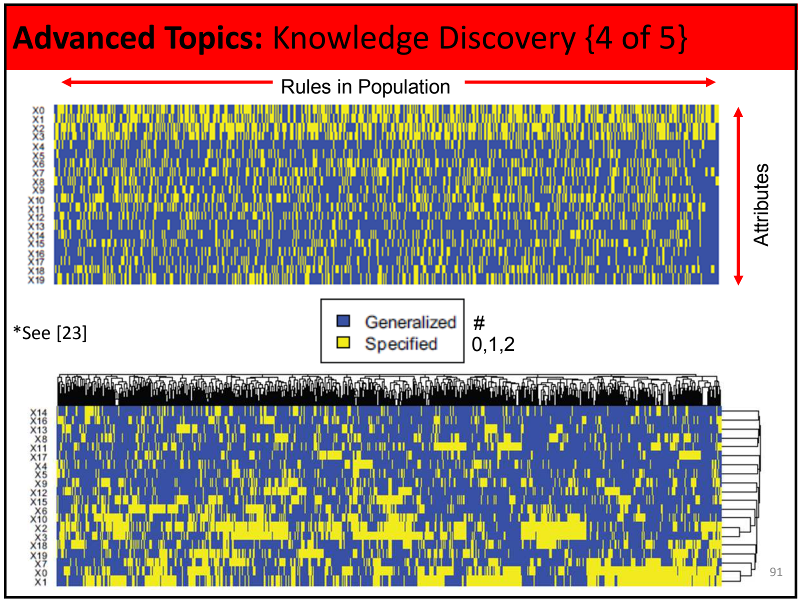
- Attributes: 20
  - Predictive: 4
  - Non-Predictive: 16
- Heritability = 0.4
- MAF = 0.2
- Sample Size = 1600
- Testing Accuracy = 0.70 (*p* = 0.001)
- See [23]

Individual Attributes

Attribute	SpS	p-value	AWSpS	p-value
X0	10885	0.001*	7589.49	0.001*
X1	11359	0.001*	7936.43	0.001*
X2	10569	0.001*	7369.84	0.001*
X3	10150	0.001*	7114.25	0.001*
X4	3863	0.999	2482.56	0.888
X5	3240	1	2090.05	1
X6	5217	0.737	3446.47	0.18
X7	5484	0.915	3647.67	0.336
X8	4429	0.95	2927.85	0.482
X9	5334	0.985	3569.25	0.484
X10	5907	0.414	3948.81	0.04*
X11	5725	0.414	3933.61	0.037*
X12	5273	1	3518.87	0.761
X13	4443	1	2854.43	0.996
X14	3709	1	2391.91	0.978

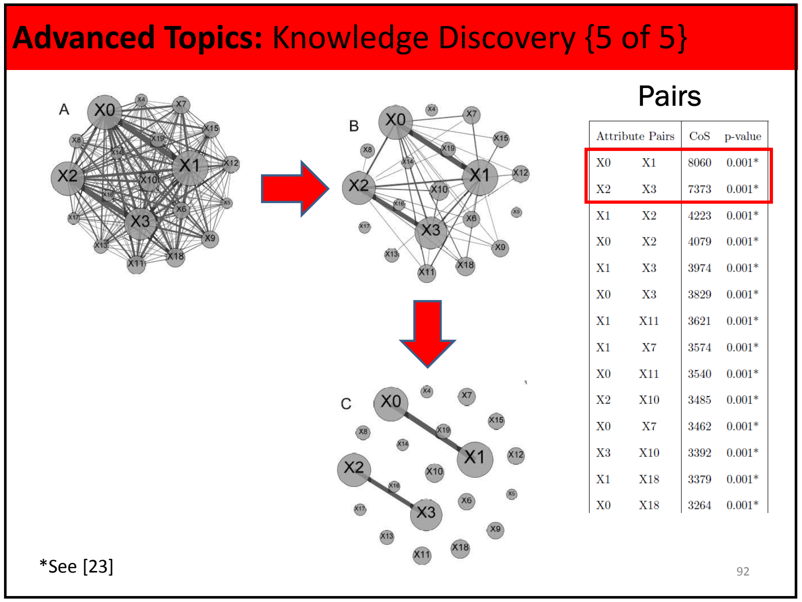
Pairs

Attribute Pairs		CoS	p-value
X0	X1	8060	0.001*
X2	X3	7373	0.001*
X1	X2	4223	0.001*
X0	X2	4079	0.001*
X1	X3	3974	0.001*
X0	X3	3829	0.001*
X1	X11	3621	0.001*
X1	X7	3574	0.001*
X0	X11	3540	0.001*
X2	X10	3485	0.001*
X0	X7	3462	0.001*
X3	X10	3392	0.001*
X1	X18	3379	0.001*
X0	X18	3264	0.001*



Advanced Topics: Knowledge Discovery {4 of 5}

*See [23]



Advanced Topics: Knowledge Discovery {5 of 5}

Pairs

Attribute Pairs		CoS	p-value
X0	X1	8060	0.001*
X2	X3	7373	0.001*
X1	X2	4223	0.001*
X0	X2	4079	0.001*
X1	X3	3974	0.001*
X0	X3	3829	0.001*
X1	X11	3621	0.001*
X1	X7	3574	0.001*
X0	X11	3540	0.001*
X2	X10	3485	0.001*
X0	X7	3462	0.001*
X3	X10	3392	0.001*
X1	X18	3379	0.001*
X0	X18	3264	0.001*

*See [23]

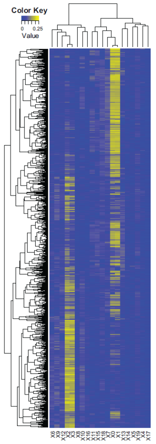
Advanced Topics: Attribute Tracking & Feedback

An extension to the LCS algorithm that allows for the explicit characterization of heterogeneity, and allows for the identification of heterogeneous subject groups.

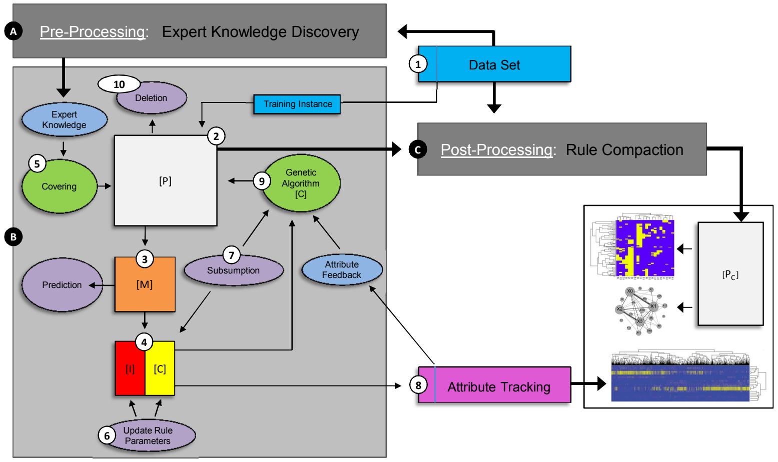
Akin to long-term memory. Experiential knowledge stored separately from the rule population that is never lost.

Relies on learning that is both incremental and supervised.

Stored knowledge may be fed back into LCS during learning.



Advanced Topics: ExSTraCS 2.0 – Shameless Plug



A Pre-Processing: Expert Knowledge Discovery

1 Data Set

Training Instance

B

10 Deletion

Expert Knowledge

2 [P]

5 Covering

9 Genetic Algorithm [C]

3 [M]

Prediction

7 Subsumption

Attribute Feedback

4 [I] [C]

6 Update Rule Parameters

C Post-Processing: Rule Compaction

$[P_C]$

8 Attribute Tracking

Advanced Topics: Rule Specificity Limit

Previous:

- Data with many attributes yields absurdly over-fit ExSTraCS rules – not sufficient pressure to generalize.
- Allows for an impractically sized search space
- Relying on P_{spec} problematic.

RSL:

- IDEA: Limit maximum rule dimensionality based on dataset characteristics (i.e. what we might have any hope of being powered to find).
- Calculate unique attribute state combinations $\psi = \epsilon^n$

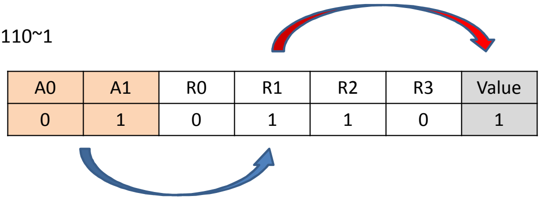
	ϵ			
n	2	3	4	5
1	2	3	4	5
2	4	9	16	25
3	8	27	64	125
4	16	81	256	625
5	32	243	1024	3125
6	64	729	4096	15625
7	128	2187	16384	78125
8	256	6561	65536	390625

Example: SNP dataset

- ϵ = 3
- *Training Instances* = 2000
- Find where : $\iota < \psi$

Advanced Topics: Multiplexer Problem

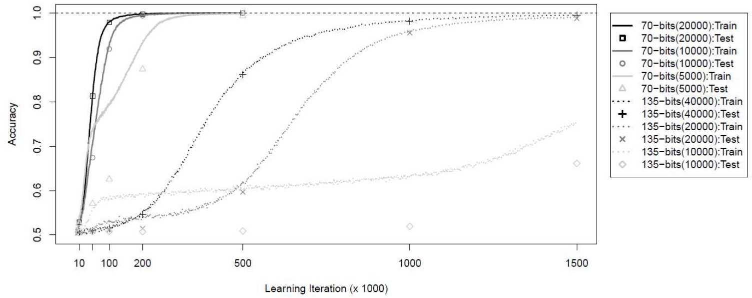
- ❖ Scalability: (3,6,11,20,37,70…-bit Multiplexer)
- ❖ Boolean 6-Multiplexer: 64 unique instances
- ❖ Task: Decode a 2 bit address and return the value of the correspondent binary data register.
- ❖ Example: 010110~1



A0	A1	R0	R1	R2	R3	Value
0	1	0	1	1	0	1

x	Address Bits	Order of Interaction	Heterogeneous Combinations	Unique Instances	Optimal Rules [O]
6-bit	2	3	4	64	8
11-bit	3	4	8	2048	16
20-bit	4	5	16	1.05×10^6	32
37-bit	5	6	32	1.37×10^{11}	64
70-bit	6	7	64	1.18×10^{21}	128
135-bit	7	8	128	4.36×10^{40}	256

Advanced Topics: ExSTraCS Results – 70 & 135-bit Multiplexer Problem



- 135-bit multiplexer problem solved indirectly in [26].
- First report of an algorithm solving this problem directly.

What is Learning?

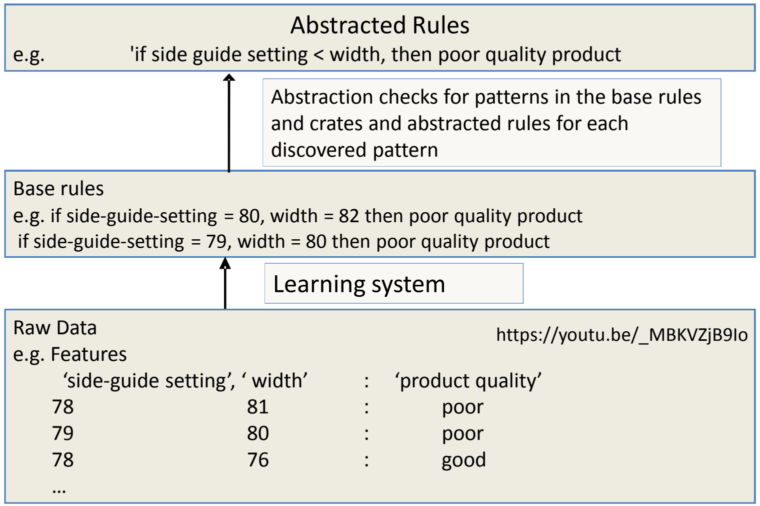
How to tie a bow:

CF Hypotheses

- Different problems in the same domain are likely to contain common patterns
- Patterns in one domain are useful in related domains

This Statement
is False

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

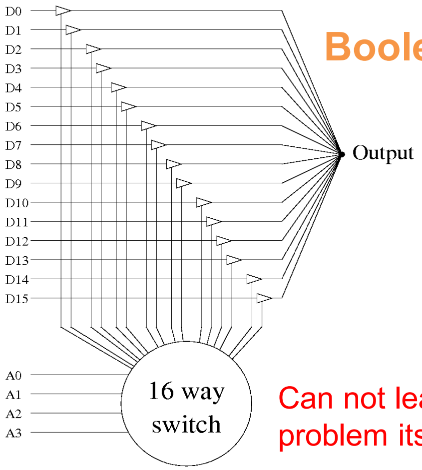
Base rules
e.g. if side-guide-setting = 80, width = 82 then poor quality product
if side-guide-setting = 79, width = 80 then poor quality product

Learning system

Raw Data
e.g. Features

https://youtu.be/_MBKVZjB9Io

‘side-guide setting’,	‘ width’	:	‘product quality’
78	81	:	poor
79	80	:	poor
78	76	:	good
…			



Boolean Multiplexer

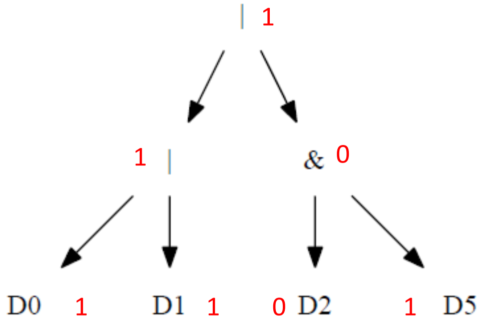
Length $n = k + 2^k$

Humans need to know:
- Recognise Patterns
- Localisation
- Binary numbers (powers and sums)
- Addressing
- Returning a value

Can not learn these within the problem itself.

0	1	1	1	#	#	#	#	#	#	#	0	#	#	#	#	#	#	#	#	#	0

Evaluating a Code Fragment



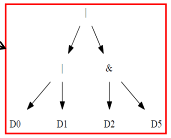
110101

D0	D1	D2	D3	D4	D5
1	1	0	1	0	1

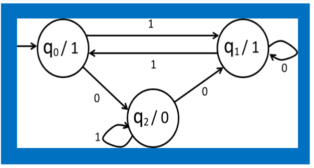
CF Systems

- XCSCFC – CF in the condition part
- XCSCFA – CF in the action part
- XCSRCFA – Real coded, CF in the action part
- XCSSMA - Replaced the static binary action with a Moore state machine

<table>
<tr><th colspan="6">Condition</th><th>Action</th></tr>
<tr><td>D0D0~|</td><td>D0D5d~</td><td>D1D4r~</td><td>D0D0~|</td><td>D0D0~|</td><td>D0D0~|</td><td>0</td></tr>
<tr><td>D0D0~|</td><td>D0D0~|</td><td>D0D0~|</td><td>D0</td><td>D1D4d</td><td>D4</td><td>1</td></tr>
<tr><td>D0D0~|</td><td>D0D5d~</td><td>D5D1&</td><td>D0D0~|</td><td>D0D0~|</td><td>D0D0~|</td><td>0</td></tr>
<tr><td>D3D0rD5D1dr</td><td>D0D0~|</td><td>D0D0~|</td><td>D0D0~|</td><td>D0D0~|</td><td>D0D0~|</td><td>1</td></tr>
<tr><td>D0D1dD0D4d&</td><td>D0D0~|</td><td>D0D1|D2D5&|</td><td>D3D1&~</td><td>D0D0~|</td><td>D0D0~|</td><td>0</td></tr>
<tr><td colspan="6">...</td><td>...</td></tr>
</table>



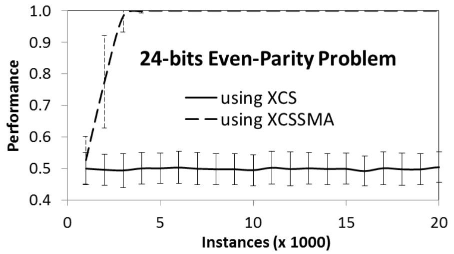
XCS with State-Machine Actions

Condition	Action
1 0 # 0 1 #	

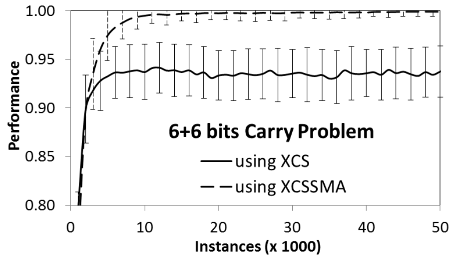
Input Message	1	0	1	0	1	0
Reached State	q_1	q_1	q_0	q_2	q_2	q_1

The ending state for this input message is q_1, therefore action value = output of state q_1 = 1.

General Solutions to Problems

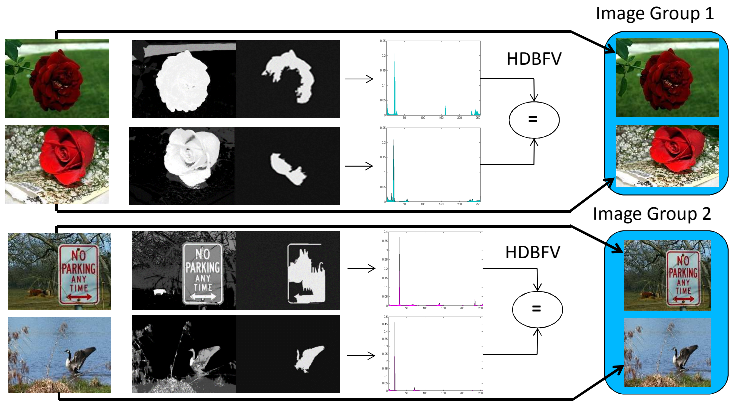


The classifier rule produced by XCSSMA can solve any n-bits even-parity problem



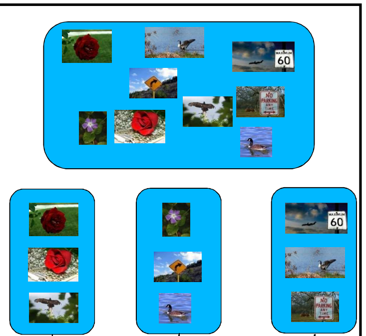
The classifier rule produced by XCSSMA can solve any n+n bits carry-one problem.

Segregating Images Based on Feature Composition

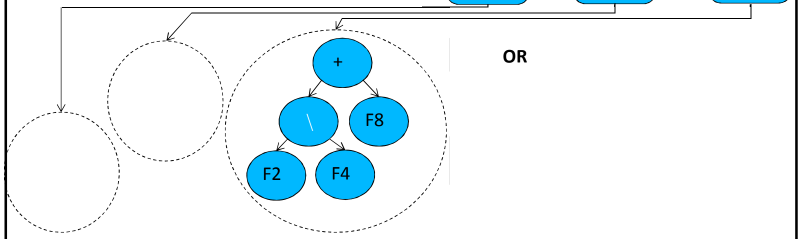


HDBFV: Histogram Distance b/w Feature Vectors

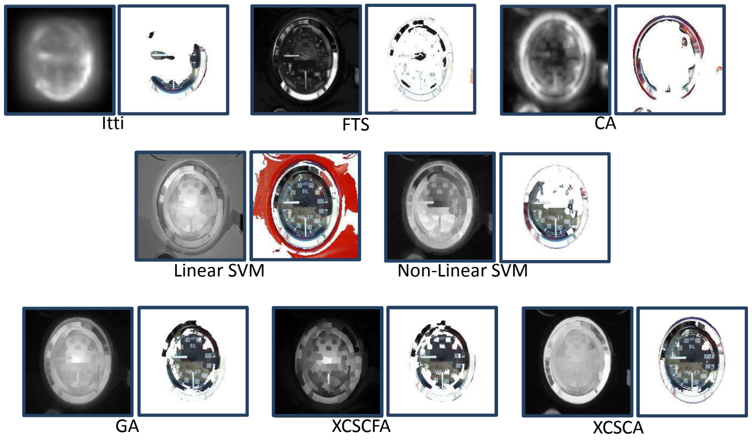
What's Required?



- Learning multiple image dependent solutions for both weighting and function based combination.

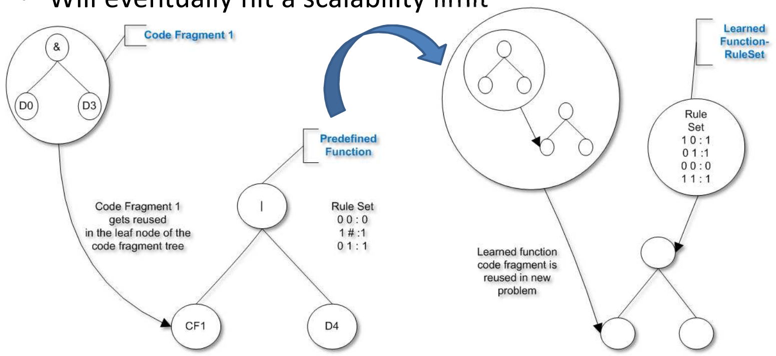


Visual Comparison

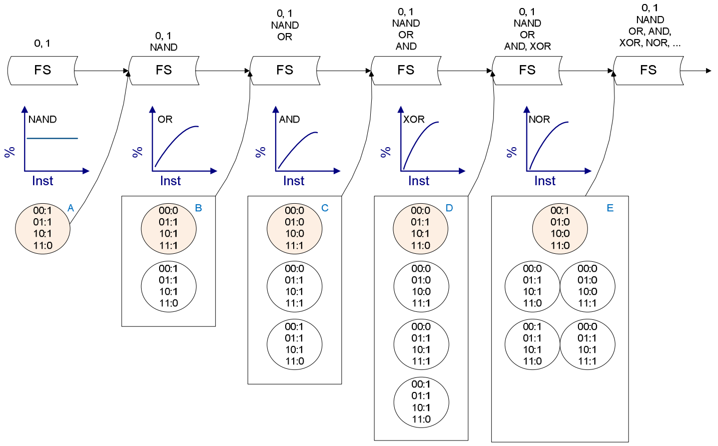


CF - Limitations

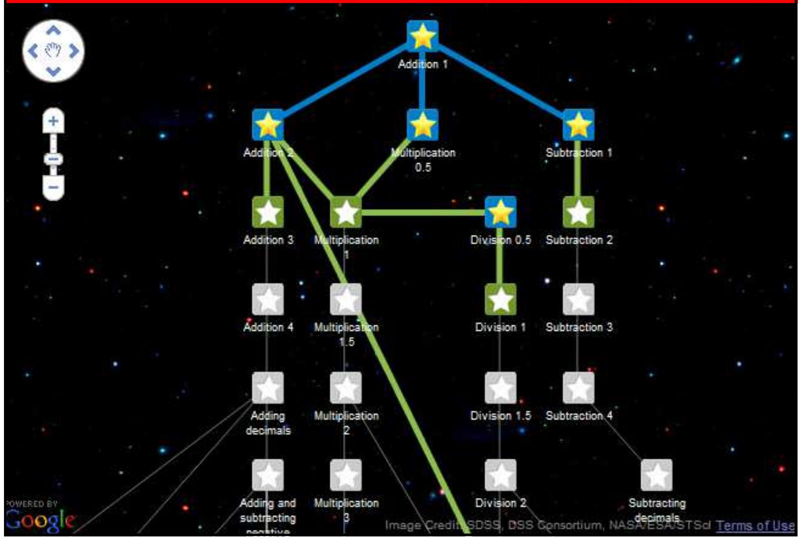
- Only replaces leaf nodes with CFs
- Does not change inner nodes
- Will eventually hit a scalability limit



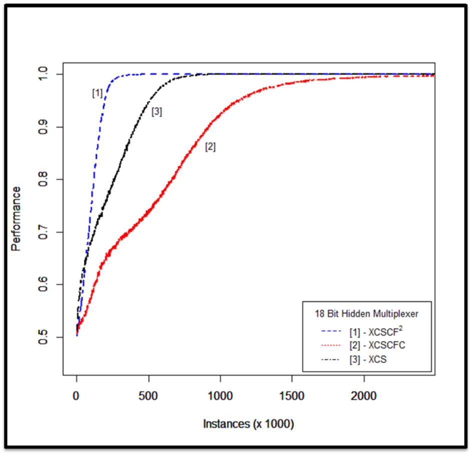
Rulesets as functions: XCSCF2



User must determine order of problems



Boolean – Parity – 6Mux - HMux



Isidro M. Alvarez -
isidro.alvarez@ecs.vuw.ac.nz

Resources – Additional Information

- Additional Information :
  - Keep up to date with the latest LCS research
  - Get in contact with an LCS researcher
  - Contribute to the LCS community research and discussions.
- Active Websites:
  - GBML Central - http://gbml.org/
  - Illinois GA Lab – http://www.illigal.org
- LCS Researcher Webpages:
  - Urbanowicz, Ryan - http://www.ryanurbanowicz.com/
  - Browne, Will - http://ecs.victoria.ac.nz/Main/WillBrowne
  - Lanzi, Pier Luca - http://www.pierlucalanzi.net/
  - Wilson, Stewart - https://www.eskimo.com/~wilson/
  - Bacardit, Jaume - http://homepages.cs.ncl.ac.uk/jaume.bacardit/
  - Holmes, John - http://www.med.upenn.edu/apps/faculty/index.php/g359/c1807/p19936
  - Kovacs, Tim - http://www.cs.bris.ac.uk/home/kovacs/
  - Bull, Larry - http://www.cems.uwe.ac.uk/~lbull/
- International Workshop Learning Classifier Systems (IWLCS) - held annually at GECCO
  - Renamed for GECCO ‘15 – Evolutionary Rule-based Machine Learning
- Other:
  - Mailing List:: Yahoo Group: lcs-and-gbml @ Yahoo
  - Proceedings of IWLCS
  - Annual Special Issue of Learning Classifier Systems published by Evolutionary Intelligence
    - NEW ISSUE THEME: 20 Years of XCS!!! – Dedicated to Stewart Wilson

Resources - Software

- Educational LCS (eLCS) – in Python.
  - http://sourceforge.net/projects/educationallcs/
  - Simple Michigan-style LCS for learning how they work and how they are implemented.
  - Code intended to be paired with first LCS introductory textbook by Brown/Urbanowicz.
- ExSTraCS 2.0 – Extended Supervised Learning LCS – in Python
  - http://sourceforge.net/projects/exstracs/
  - For prediction, classification, data mining, knowledge discovery in complex, noisy, epistatic, or heterogeneous problems.
- BioHEL – Bioinformatics-oriented Hierarchical Evolutionary Learning – in C++
  - http://ico2s.org/software/biohel.html
  - GAssist also available through this link.
- XCS & ACS (by Butz in C and Java) & XCSLib (XCS and XCSF) (by Lanzi in C++)
  - http://www.illigal.org
- XCSF with function approximation visualization – in Java
  - http://medal.cs.umsl.edu/files/XCSFJava1.1.zip
- EpiXCS

Resources – LCS Review Papers & Books

- Select Review Papers:
  - Bull, Larry. "A brief history of learning classifier systems: from CS-1 to XCS and its variants." *Evolutionary Intelligence* (2015): 1-16.
  - Bacardit, Jaume, and Xavier Llorà. "Large-scale data mining using genetics-based machine learning." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 3.1 (2013): 37-61.
  - Urbanowicz, Ryan J., and Jason H. Moore. "Learning classifier systems: a complete introduction, review, and roadmap." *Journal of Artificial Evolution and Applications* 2009 (2009): 1.
  - Sigaud, Olivier, and Stewart W. Wilson. "Learning classifier systems: a survey." *Soft Computing* 11.11 (2007): 1065-1078.
  - Holland, John H., et al. "What is a learning classifier system?." *Learning Classifier Systems*. Springer Berlin Heidelberg, 2000. 3-32.
  - Lanzi, Pier Luca, and Rick L. Riolo. "A roadmap to the last decade of learning classifier system research (from 1989 to 1999)." *Learning Classifier Systems*. Springer Berlin Heidelberg, 2000. 33-61.
- Books:
  - **NOTE:** Brown & Urbanowicz are preparing an Introductory LCS Textbook – hopefully available this year. (Springer)
  - Drugowitsch, J., (2008) Design and Analysis of Learning Classifier Systems: A Probabilistic Approach. Springer-Verlag.
  - Bull, L., Bernado-Mansilla, E., Holmes, J. (Eds.) (2008) Learning Classifier Systems in Data Mining. Springer
  - Butz, M (2006) Rule-based evolutionary online learning systems: A principled approach to LCS analysis and design. Studies in Fuzziness and Soft Computing Series, Springer.
  - Bull, L., Kovacs, T. (Eds.) (2005) Foundations of learning classifier systems. Springer.
  - Kovacs, T. (2004) Strength or accuracy: Credit assignment in learning classifier systems. Springer.
  - Butz, M. (2002) Anticipatory learning classifier systems. Kluwer Academic Publishers.
  - Lanzi, P.L., Stolzmann, W., Wilson, S., (Eds.) (2000). Learning classifier systems: From foundations to applications (LNAI 1813). Springer.
  - Holland, J. H. (1975). Adaptation in natural and artificial systems. University of Michigan Press.

Conclusion

- What and Why
  - Many branches of RBML, e.g. ARM, AIS, LCS
  - Powerful, human interpretable, learning algorithms
- Driving Mechanisms
  - Discovery
  - Learning
- How?
  - LCS Algorithm Walk-Through
  - Flexible and robust methods developed
- Multiple styles
- Advanced methods: solutions to complex & real-world problems
- Many Resources Available

References {1 of 4}

1) Urbanowicz, Ryan John, et al. "Role of genetic heterogeneity and epistasis in bladder cancer susceptibility and outcome: a learning classifier system approach." *Journal of the American Medical Informatics Association* (2013)
2) Holland, J., and J. Reitman. "Cognitive systems based on adaptive agents."*Pattern-directed inference systems* (1978).
3) Smith, Stephen Frederick. "A learning system based on genetic adaptive algorithms." (1980).
4) Booker, Lashon Bernard. "Intelligent behavior as an adaptation to the task environment, University of Michigan." *Ann Arbor, MI* (1982).
5) Holland, J. "Properties of the Bucket brigade.*" In Proceedings of the 1st International Conference on Genetic Algorithms*, 1-7 (1985)
6) Frey, Peter W., and David J. Slate. "Letter recognition using Holland-style adaptive classifiers." *Machine Learning* 6.2 (1991): 161-182.
7) Riolo, Rick L. "Lookahead planning and latent learning in a classifier system."*Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*. MIT Press, 1991.
8) Wilson, Stewart W. "ZCS: A zeroth level classifier system." *Evolutionary computation* 2.1 (1994): 1-18.
9) Wilson, Stewart W. "Classifier fitness based on accuracy." *Evolutionary computation* 3.2 (1995): 149-175.
10) Holmes, John H. "A genetics-based machine learning approach to knowledge discovery in clinical data." *Proceedings of the AMIA Annual Fall Symposium*. American Medical Informatics Association, 1996.
11) Stolzmann, Wolfgang. "An introduction to anticipatory classifier systems."*Learning Classifier Systems*. Springer Berlin Heidelberg, 2000. 175-194.

References {2 of 4}

12) Wilson, Stewart W. "Classifiers that approximate functions." *Natural Computing*1.2-3 (2002): 211-234.
13) Kovacs, Tim. "A comparison of strength and accuracy-based fitness in learning classifier systems." *School of Computer Science, University of Birmingham, Birmingham, UK* (2002).
14) Kovacs, Tim. "What should a classifier system learn and how should we measure it?." *Soft Computing* 6.3-4 (2002): 171-182.
15) Bernadó-Mansilla, Ester, and Josep M. Garrell-Guiu. "Accuracy-based learning classifier systems: models, analysis and applications to classification tasks."*Evolutionary Computation* 11.3 (2003): 209-238.
16) Bull, Larry. "A simple accuracy-based learning classifier system." *Learning Classifier Systems Group Technical Report UWELCSG03-005, University of the West of England, Bristol, UK* (2003).
17) Peñarroya, Jaume Bacardit. *Pittsburgh genetic-based machine learning in the data mining era: representations, generalization, and run-time*. Diss. Universitat Ramon Llull, 2004.
18) Bacardit, Jaume, Edmund K. Burke, and Natalio Krasnogor. "Improving the scalability of rule-based evolutionary learning." *Memetic Computing* 1.1 (2009): 55-67.
19) Holmes, John H., and Jennifer A. Sager. "The EpiXCS workbench: a tool for experimentation and visualization." *Learning Classifier Systems*. Springer Berlin Heidelberg, 2007. 333-344.
20) Butz, Martin V. "Documentation of XCSFJava 1.1 plus visualization." *MEDAL Report* 2007008 (2007).
21) Lanzi, Pier Luca, and Daniele Loiacono. "Classifier systems that compute action mappings." *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007.

References {3 of 4}

22) Franco, María A., Natalio Krasnogor, and Jaume Bacardit. "Speeding up the evaluation of evolutionary learning systems using GPGPUs." *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010.
23) Urbanowicz, Ryan J., Ambrose Granizo-Mackenzie, and Jason H. Moore. "An analysis pipeline with statistical and visualization-guided knowledge discovery for michigan-style learning classifier systems." *Computational Intelligence Magazine, IEEE* 7.4 (2012): 35-45.
24) Urbanowicz, Ryan, Ambrose Granizo-Mackenzie, and Jason Moore. "Instance-linked attribute tracking and feedback for michigan-style supervised learning classifier systems." *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. ACM, 2012.
25) Urbanowicz, Ryan J., Delaney Granizo-Mackenzie, and Jason H. Moore. "Using expert knowledge to guide covering and mutation in a michigan style learning classifier system to detect epistasis and heterogeneity." *Parallel Problem Solving from Nature-PPSN XII*. Springer Berlin Heidelberg, 2012. 266-275.
26) Iqbal, Muhammad, Will N. Browne, and Mengjie Zhang. "Extending learning classifier system with cyclic graphs for scalability on complex, large-scale boolean problems." *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. ACM, 2013.
27) Bacardit, Jaume, and Xavier Llorà. "Large-scale data mining using genetics-based machine learning." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 3.1 (2013): 37-61.
28) Urbanowicz, Ryan J., and Jason H. Moore. "ExSTraCS 2.0: description and evaluation of a scalable learning classifier system." *Evolutionary Intelligence*(2015): 1-28.

References {4 of 4}

29) Urbanowicz, Ryan J., and Jason H. Moore. "The application of michigan-style learning classifier systems to address genetic heterogeneity and epistasis in association studies." *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010.
30) Wilson, Stewart W. "Get real! XCS with continuous-valued inputs." *Learning Classifier Systems*. Springer Berlin Heidelberg, 2000. 209-219.
31) Stone, Christopher, and Larry Bull. "For real! XCS with continuous-valued inputs." *Evolutionary Computation* 11.3 (2003): 299-336.
32) Llora, Xavier, and Josep Maria Garrell i Guiu. "Coevolving Different Knowledge Representations With Fine-grained Parallel Learning Classifier Systems."*GECCO*. 2002.
33) Bacardit, Jaume, and Natalio Krasnogor. "A mixed discrete-continuous attribute list representation for large scale classification domains." *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 2009.
34) Goldberg, David E. "E. 1989. Genetic Algorithms in Search, Optimization, and Machine Learning." *Reading: Addison-Wesley* (1990).
35) Urbanowicz, Ryan J., and Jason H. Moore. "Learning classifier systems: a complete introduction, review, and roadmap." *Journal of Artificial Evolution and Applications* 2009 (2009): 1.
36) Bacardit, Jaume, et al. "Speeding-up pittsburgh learning classifier systems: Modeling time and accuracy." *Parallel Problem Solving from Nature-PPSN VIII*. Springer Berlin Heidelberg, 2004.