Introducing Learning Classifier Systems: Rules that Capture Complexity

Ryan J. Urbanowicz
University of Pennsylvania
Philadelphia, PA, USA

ryanurb@upenn.edu
www.ryanurbanowicz.com

Danilo Vasconcellos Vargas
Kyushu University
Japan

vargas@inf.kyushu-u.ac.jp
http://itslab.csce.kyushu-u.ac.jp/~vargas/index.php

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan © 2018 Copyright is held by the owner/author(s). ACM ISBN 978-1-4503-5764-7/18/07. https://doi.org/10.1145/3205651.3207869

Instructors

- **Ryan Urbanowicz** is a 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 development and application of advanced machine learning methods for complex, heterogeneous problems in bioinformatics, genetics, and epidemiology. He has been an active contributor to the rule-based machine learning and learning classifier system community since 2009.

- **Danilo Vasconcellos Vargas** is an Assistant Professor at the Faculty of Information Science and Electrical Engineering, Kyushu University, Japan. He received the B.Eng. degree in computer engineering from the University of São Paulo, São Paulo, Brazil, in 2009, and both the M.Eng. and Ph.D. degree from Kyushu University, Fukuoka, Japan, in 2014 and 2016 respectively. His current research interests focus on general learning systems which include research in evolutionary algorithms, neural networks, Learning Classifier Systems (LCS) and their applications.

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
  - Rule Compaction
- Michigan vs. Pittsburgh-style
- Advanced Topics
- Resources

Multiplexer Benchmark Problem

- *"Multiplexer functions have long been identified by researchers as functions that often pose difficulties for paradigms for machine learning, artificial intelligence, neural nets, and classifier systems."* – [John Koza - Foundations of Genetic Algorithms, 1991]
- Multiplexer Problem Characteristics:
  – Multivariate, non-linearity, epistasis, heterogeneity/latent class.
- TO SOLVE: Any Multiplexer
  – No single feature has any association with endpoint
  – Only a certain subset of features are predictive for a given individual belonging to an underlying subgroup (i.e. latent class)

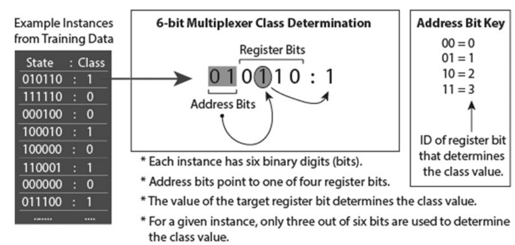
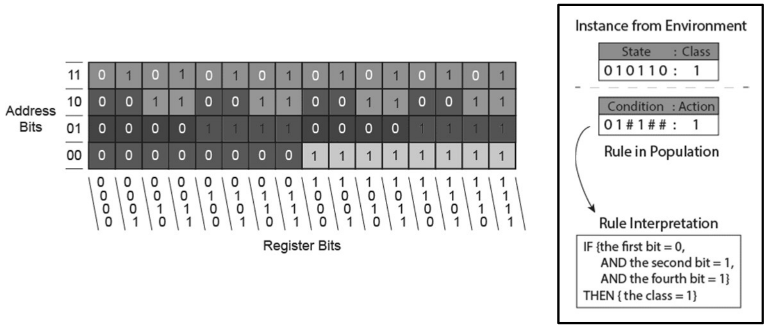


*Image adapted from [37]

6-bit Multiplexer Benchmark Problem

- **Epistatic Non-linearity** – A dependence between features that impacts outcome in a non-linear, non-additive fashion.
- **Heterogeneneous Pattern of Association** – Independent features or groups of features impact outcome within different subsets of training/testing instances.



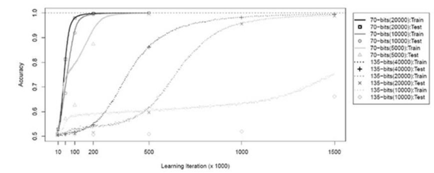
*Images adapted from [37]

Solving the 135-bit Multiplexer

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

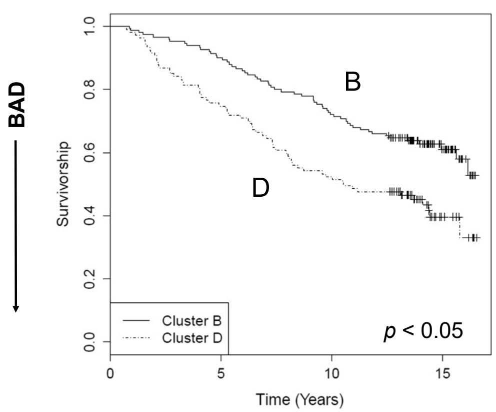
- TO SOLVE: 135-bit Multiplexer
  - All 135 features are predictive in at least some subset of the dataset.
  - Non-RBML approaches would need to include all 135 attributes together in a single model properly capturing underlying epistasis and heterogeneity.
- Few ML algorithms can make the claim that they can solve even the 6 or 11-bit multiplexer problems, let alone the 135-bit multiplexer.

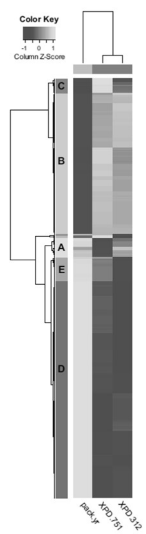
ExSTraCS



*Images adapted from [28]

Bladder Cancer Study: Clinical Variable Analysis-Survivorship





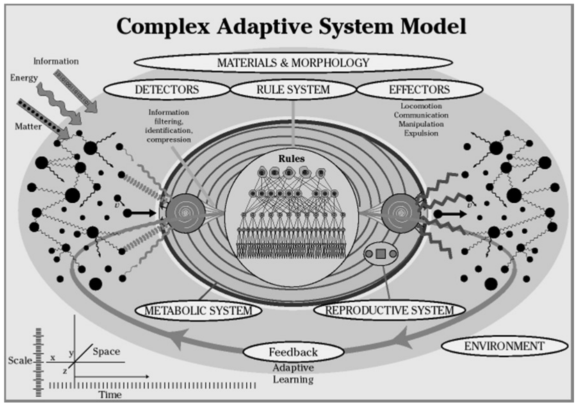
*Images adapted from [1]

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.*

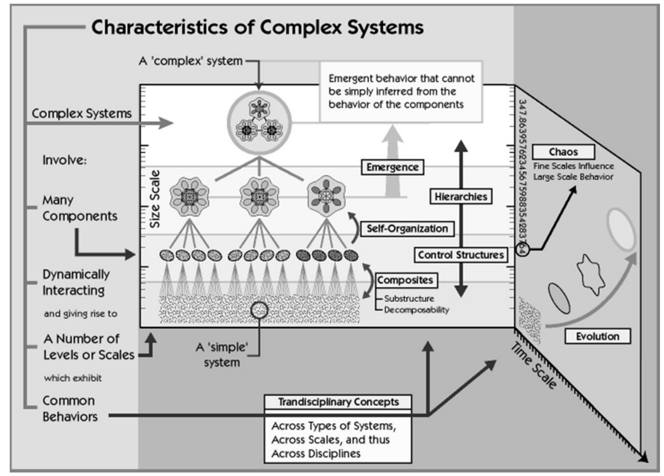
Introduction: Complex Adaptive Systems (1 of 3)



Courtesy of Marshall Clemens and New England Complex Systems Institute.

(See Holland [39])

Introduction: Complex Adaptive Systems (2 of 3)



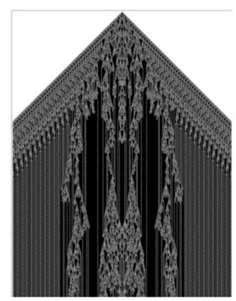
Courtesy of Marshall Clemens and New England Complex Systems Institute.

Introduction: Complex Adaptive Systems (3 of 3)

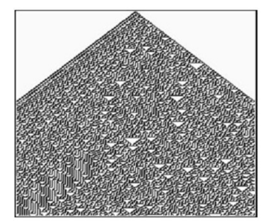
- **Multi-Agent System** - A collection of primitive components, called "agents"
- **Interaction** - Interactions among agents and between agents and their environments
- **Emergence** - Unanticipated global properties often result from the interactions
- **Adaptation** - Agents adapt their behavior to other agents and environmental constraints
- **Evolution** - As a consequence, system behavior evolves over time

Quoted from [38]

Introduction: Agent-based Modeling and ALIFE

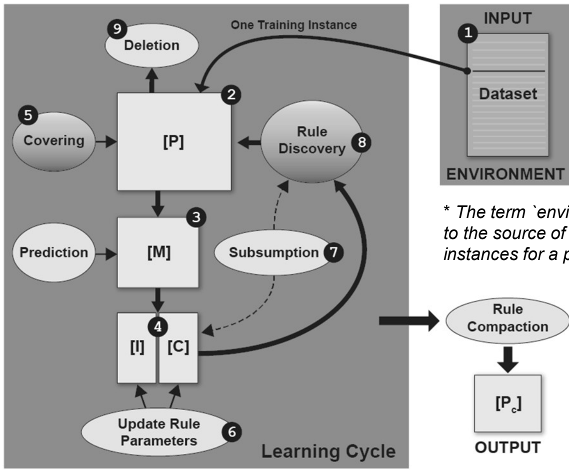


Wolfram



Welcome to the realm of simple chaotic systems!
Cellular Automata [43]

Introduction: LCS In A Nutshell – A Basic Schematic



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

*Image adapted from [37]

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 [23].
- **Implicitly Parsimonious** – Rule evolution has an implicit generalization pressure towards parsimonious rules/solutions.

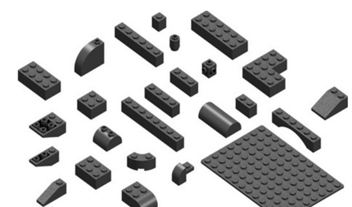
Introduction: Why LCS Algorithms? {2 of 3}

- **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 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
  - …

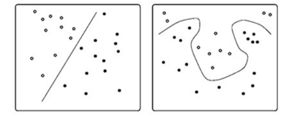
*Slide adapted from Lanzi Tutorial: GECCO 2014

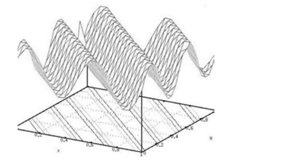
Introduction: LCS Disadvantages

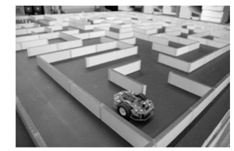
- ❖ Not widely known.
- ❖ Relatively limited software accessibility.
- ❖ Rule population interpretation and knowledge extraction can be challenging.
- ❖ Can suffer from overfitting, despite explicit and implicit pressures to generalize rules.
- ❖ Relatively little theoretical work or convergence proofs.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

Introduction: LCS Applications - General

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





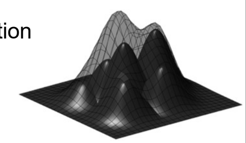


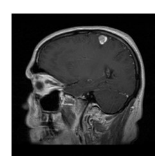
Introduction: LCS Applications – Uniquely Suited To…

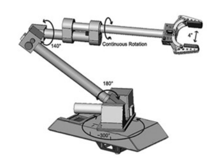
- ❖ **Uniquely Suited To Problems with…**
  - ❖ Dynamic environments
  - ❖ Perpetually novel events accompanied by large amounts of noisy or irrelevant data.
  - ❖ Continual, often real-time, requirements for actions.
  - ❖ Implicitly or inexactly defined goals.
  - ❖ Sparse payoff or reinforcement obtainable 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
Optimization
Modelling
Scheduling
Design
Routing
Clustering
Knowledge-Handling
Prediction
Visualisation
Feature Selection
Querying
Image classification
Adaptive-control
Navigation
Game-playing
Rule-Induction
Data-mining



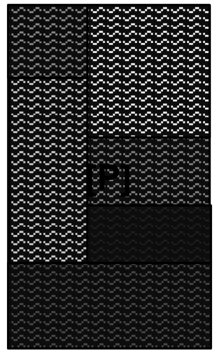




Introduction: Distinguishing Features of an LCS

- **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: 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…
  - Rule-Based Machine Learning (RBML)
  - Genetics Based Machine Learning (GBML)
  - Adaptive Agents
  - Cognitive Systems
  - Production Systems
  - Classifier System (CS, CFS)

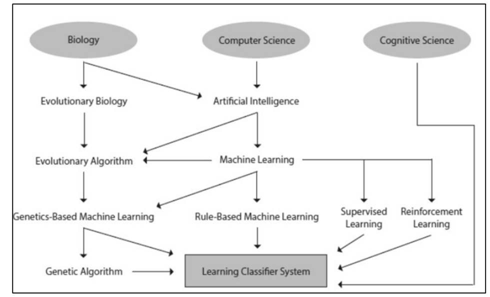


*Image adapted from [37]

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

- Pittsburgh-style algorithms introduced by Smith in Learning Systems One (LS-1) [3]

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

- 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

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].

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

- **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.

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

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

Driving Mechanisms: GA – Mutation Operator

❖ Select parent rule $r_1 = 01110001$

❖ Randomly select bit to mutate $r_1 = 01110001$

❖ Apply mutation $r_1 = 01100001$

Randomise	Generalise	Specialise	
0 → 1 or #	0 → #	# → 0 or 1	* Some LCS algorithms do not allow specialisation to a different state value (e.g. 0 → 1 or 1 → 0).
1 → 0 or #	1 → #	0 → 1	
# → 0 or 1		1 → 0	

*Image adapted from [37]

Driving Mechanisms: GA – Crossover Operator

	Single-Point Crossover	Two-Point Crossover	Uniform Crossover
Select Parents	$P_1 = 000100 : 1$ $P_2 = 011101 : 1$	$P_1 = 000100 : 1$ $P_2 = 011101 : 1$	$P_1 = 000100 : 1$ $P_2 = 011101 : 1$
Set Crossover Point(s)	$O_1 = 000100 : 1$ $O_2 = 011101 : 1$	$O_1 = 000100 : 1$ $O_2 = 011101 : 1$	$O_1 = 000100 : 1$ $O_2 = 011101 : 1$
Crossover	$O_1 = 000100 : 1$ $O_2 = 011101 : 1$	$O_1 = 000100 : 1$ $O_2 = 011101 : 1$	$O_1 = 000100 : 1$ $O_2 = 011101 : 1$
Crossover Complete in Offspring Rules	$O_1 = 000101 : 1$ $O_2 = 011100 : 1$	$O_1 = 001100 : 1$ $O_2 = 010101 : 1$	$O_1 = 001101 : 1$ $O_2 = 010100 : 1$

*Image adapted from [37]

Driving Mechanisms

Two mechanisms are primarily responsible for driving LCS algorithms.

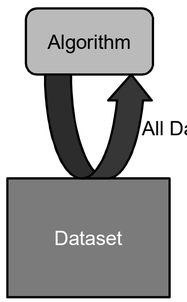
- ❖ Discovery
  - ❖ Refers to "rule discovery"
  - ❖ Traditionally performed by a genetic algorithm (GA)
  - ❖ Can use any directed method to find new rules
- ❖ Learning
  - ❖ The improvement of performance in some environment through the acquisition of knowledge resulting from experience in that environment.
  - ❖ 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

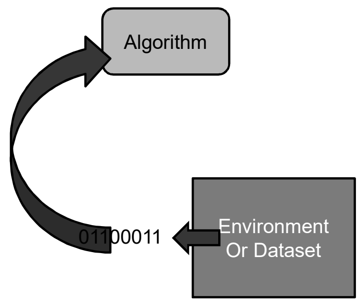
- ❖ With the advent of computers, humans have been interested in seeing how artificial 'agents' could learn. Either learning to…
  - ❖ Solve problems of value that humans find difficult to solve
  - ❖ For the curiosity of how learning can be achieved.
- ❖ Learning strategies can be divided up in a couple 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.

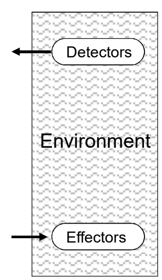
LCS Algorithm Walk-Through: Input {1 of 3}

- Input to the algorithm is often a training dataset.
- The source of input is often referred to as the 'environment'.

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

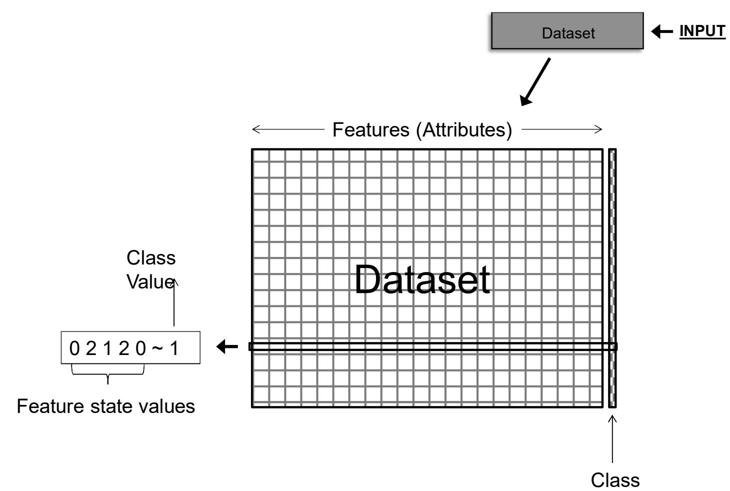
LCS Algorithm Walk-Through: Input {2 of 3}

- 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.

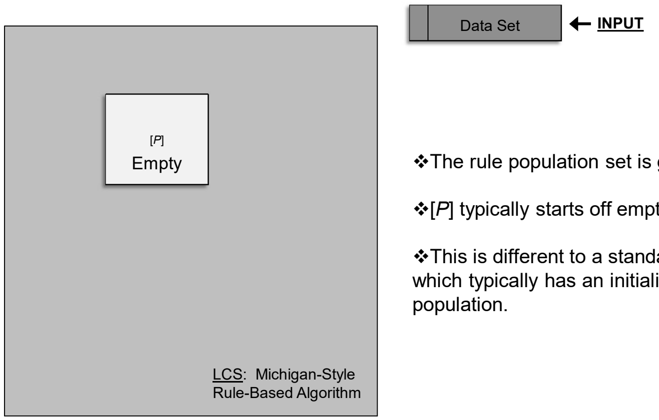


* Primarily relevant to reinforcement learning systems outside the UCS framework.

LCS Algorithm Walk-Through: Input {3 of 3}



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}

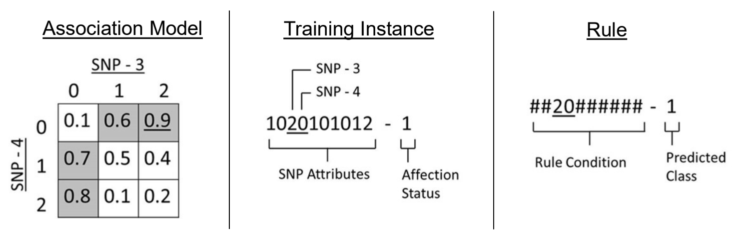
- ❖ A finite set of rules [*P*] which collectively explore the 'search space'.
- ❖ Every valid rule can be thought of as part of a candidate solution (may or may not be good)
- ❖ The space of all candidate solutions is termed the 'search space'.
- ❖ The size of the search space is determined by both 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.



LCS Algorithm Walk-Through: Rule Representation - Ternary

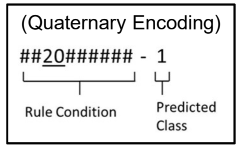
- ❖ LCSs can use many different representation schemes.
  - ❖ Also referred to as `encodings'
  - ❖ Suited to binary input or
  - ❖ Suited to real-valued inputs and so forth...
- ❖ Ternary Encoding – traditionally most commonly used
  - ❖ The ternary alphabet matches binary input
- ❖ A attribute in the condition that we don't care about is given the symbol '#' (wild card)

(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}

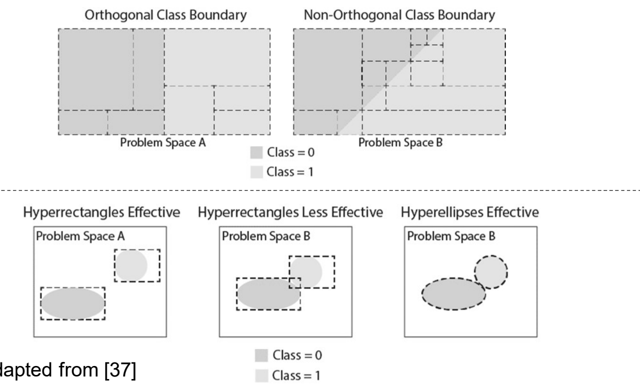
- Quaternary Encoding [29]
  - 3 possible attribute states {0,1,2} plus '#'.
  - For a specific application in genetics.



- 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}

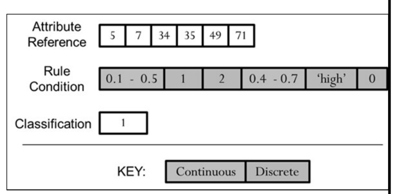
- Real-valued intervals form hyperrectangles.
- Hyperellipses may offer a more effective alternative in problems with non-orthogonal class boundaries.

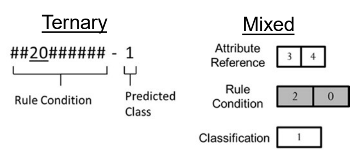


*Images adapted from [37]

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

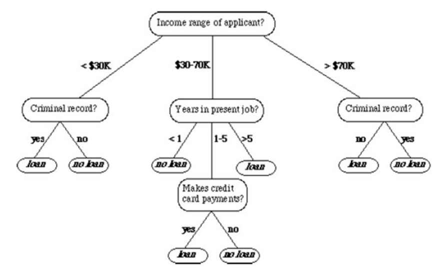
- Mixed Discrete-Continuous ALKR [28]
  - Useful for big and data with multiple attribute types
    - Discrete (Binary, Integer, String)
    - Continuous (Real-Valued)
  - Similar to ALKR (Attribute List Knowledge Representation): [Bacardit et al. 09]
  - Intervals used for continuous attributes and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



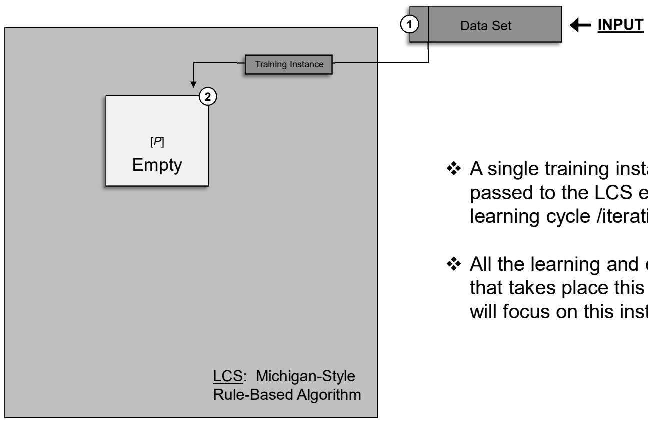


LCS Algorithm Walk-Through: Rule Representation – Other {4 of 4}

- Decision trees [32]
- Code Fragments [26]
- Artificial neural networks
- 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]

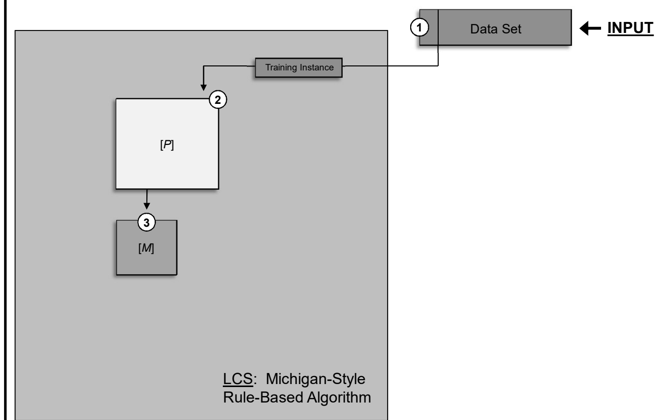


LCS Algorithm Walk-Through: Get Training Instance



- 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 {1 of 3}

[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: Matching {2 of 3}

[M]

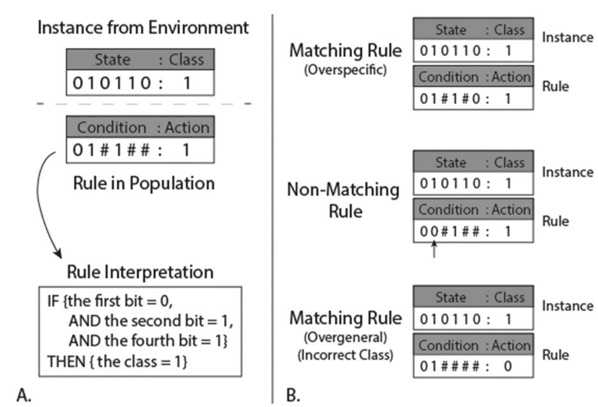


*Image adapted from [37]

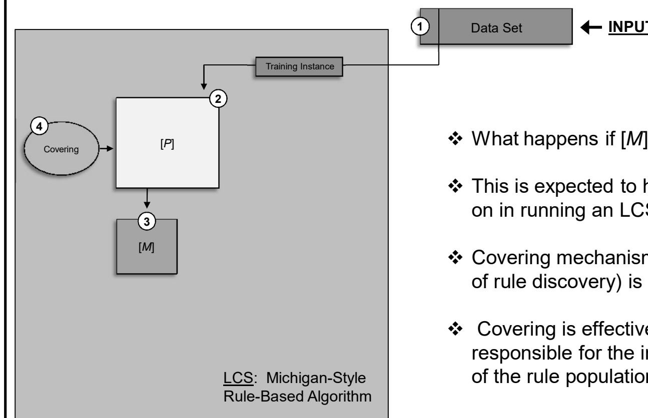
LCS Algorithm Walk-Through:
Matching {3 of 3}

[M]

Rule Representation	Example Instance	Example Matching Rule	Example Non-Matching Rule
Ternary (state values 0 or 1)	101000 : 0	##10#0 : 0	0####0 : 0
	001110 : 1	0#1### : 0	010### : 1
Integer (e.g. state values 0 - 5)	0,5,2,1,3,3 : 0	#,5,2,#,#,3 : 0	#,#,3,#,#,# : 0
	5,5,0,1,1,1 : 1	5,#,#,#,#,1 : 1	3,1,0,#,1,# : 1
Real Lower-Upper Bound (e.g. state values 0.0 - 1.0)	0.1,0.7,0.5,0.9 : 0	u #,0.7,0.6,# : 1 l #,0.5,0.4,# : 1	u 0.1,#,1.0,1.0 : 0 l 0.0,#,0.6,0.8 : 0
	0.4,0.8,0.2,0.2 : 1	u 0.6,#,0.3,# : 1 l 0.3,#,0.2,# : 1	u #,#,0.9,# : 1 l #,#,0.6,# : 1
Real Center-Spread (e.g. state values 0.0 - 1.0)	0.1,0.7,0.5,0.9 : 0	c #,0.6,0.5,# : 0 s #,0.2,0.1,# : 0	c 0.5,#,0.9,0.3 : 0 s 0.1,#,0.2,0.4 : 0
	0.4,0.8,0.2,0.2 : 1	c 0.4,#,0.3,# : 1 s 0.2,#,0.5,# : 1	c #,#,0.5,# : 1 s #,#,0.1,# : 1

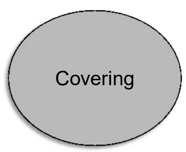
*Image adapted from [37]

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

↓

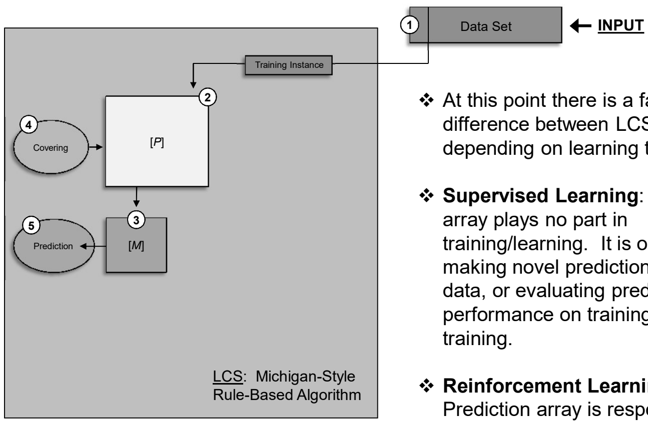
0 # 1 2 # ~ 1

(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 3}

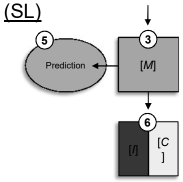


- At this point there is a fairly big difference between LCS operation depending on learning type.
- **Supervised Learning**: Prediction array plays no part in training/learning. It is only useful in making novel predictions on unseen data, or evaluating predictive performance on training data during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

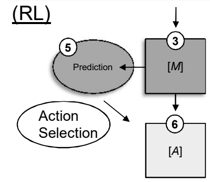
LCS Algorithm Walk-Through:
Prediction Array {2 of 3}

- 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.

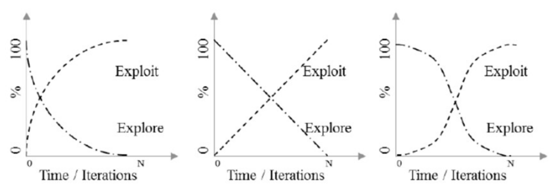
Supervised Learning (SL)



Reinforcement Learning (RL)

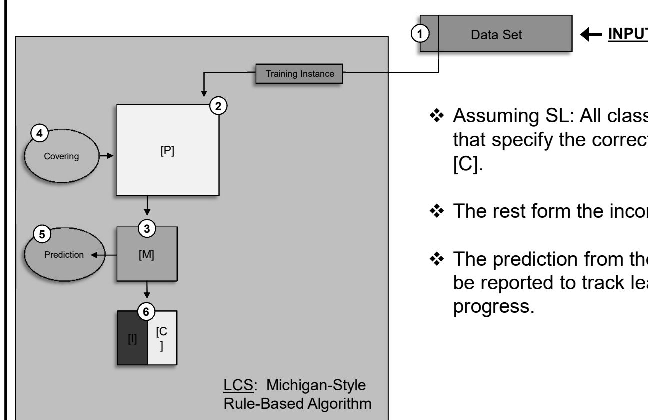


LCS Algorithm Walk-Through:
RL - Explore vs. Exploit



- One of the biggest problems in evolutionary computation…
  - When to exploit the knowledge that is being learned (i.e. vote for action)?
  - When to explore to learn new knowledge (i.e. random action)?
- 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

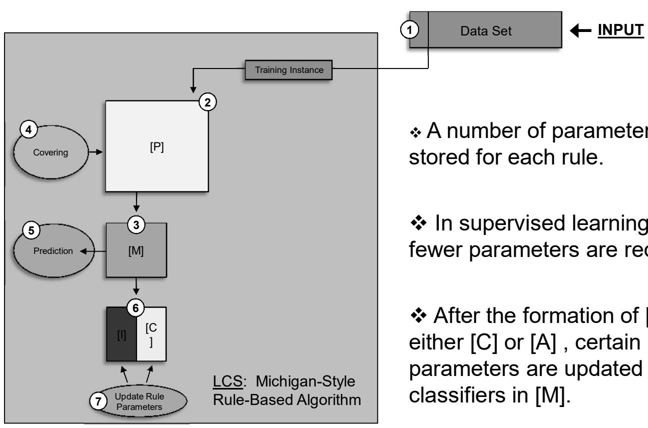
1 2 # ~ 0

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.
- ❖ 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

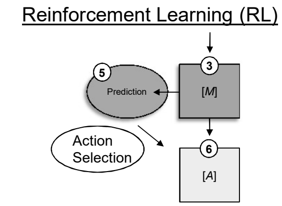
$$acc = \frac{\text{number of correct classifications}}{\text{experience}}$$

- 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

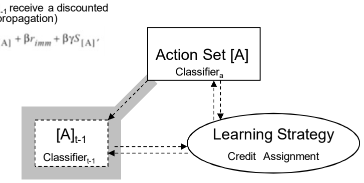
LCS Algorithm Walk-Through: Credit Assignment for Reinforcement Learning



- LCS algorithms were originally all designed with RL in mind.
- Credit traditionally took the form of classifier strength
  - The cumulative credit coming from reward feedback from the environment
  - This reflects the reward the system can expect if that rule is fired.
- Two examples of strength-based credit assignment/fitness:
  - ZCS – Zeroth-Level Classifier System [8]
    - Implicit Bucket Brigade back-propagation of strength (deferred reward)
    - Fraction (β) of strength of all rules in [A] is placed in a common 'bucket'.
    - If an immediate reward (r_{imm}) is received from environment all rules in [A] add (β r_{imm}/ [A])
    - Classifiers in the action set of the previous time-step $[A]_{-1}$ receive a discounted (γ) distribution of the strength put in the 'bucket' (back-propagation)
    - Total strength of members of [A] $S_{[A]} \leftarrow S_{[A]} - \beta S_{[A]} + \beta r_{imm} + \beta\gamma S_{[A]'}$
  - MCS – Minimal Classifying System [16]
    - Widrow-Hoff delta rule with 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

$$f_j <- f_j + \beta\,((P / \|[A]\|) - f_j)$$

    - Also applies fitness sharing….



LCS Algorithm Walk-Through: Fitness Sharing

- Fitness sharing takes the strength/payoff and updates a fitness so that the strength of a classifier is considered relative to the strengths of other classifiers in the action set.
- This pressures the classifiers with the best strength relative to their niche to have the highest fitness. This helps eliminate the takeover effect of 'strong' classifiers from one particular niche.
- Niche: A set of environmental states each of which is matched by approximately the same set of classifiers.
- We will detail fitness sharing in the context of XCS and accuracy-based fitness.

LCS Algorithm Walk-Through: Why not Strength vs. Accuracy-based Fitness in RL?

- Different niches of the environment usually have different payoff levels.
- In fitness sharing, a 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
- ZCS → XCS : "Wilson's intuition was the prediction should estimate how much reward might result from a certain action but that the evolution learning should be focused on the most reliable classifiers, that is, classifiers that give a more precise (accurate) prediction)"

LCS Algorithm Walk-Through:
XCS Accuracy-Based Fitness + Fitness Sharing

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

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

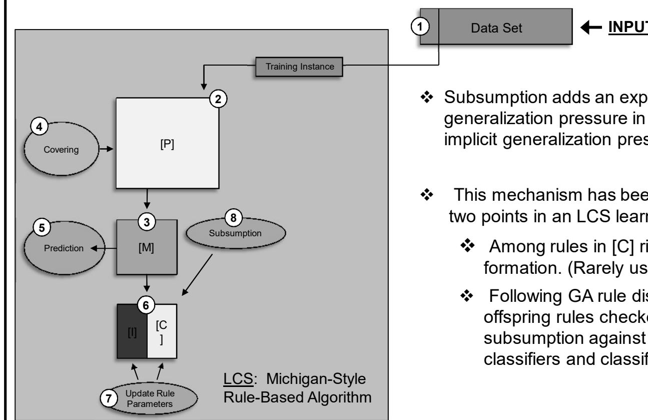
$$\varepsilon \leftarrow \varepsilon + \beta(|R - p| - \varepsilon),$$

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

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

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

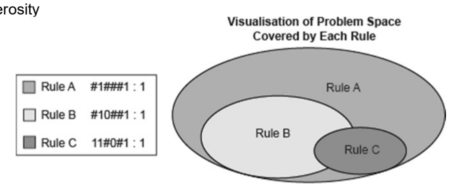
LCS Algorithm Walk-Through:
Subsumption {1 of 2}



- ❖ Subsumption adds an explicit rule generalization pressure in addition to the implicit generalization pressure.
- ❖ 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.
- ❖ 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 = General rule (A) absorbs a more specific one (B)
  - ❖ Increases rule numerosity



Requirements for Subsumption: (e.g. Can Rule A subsume Rule B?)
(1) Rules A and B have same action/class.
(2) Rule A covers Rule B completely.
(3) Rule A is accurate (i.e. error is below accuracy threshold).

*Image adapted from [37]

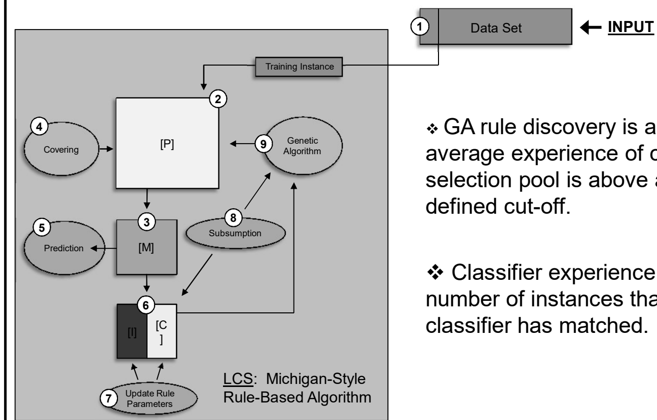
LCS Algorithm Walk-Through:
Numerosity {1 of 2}

- ❖ 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



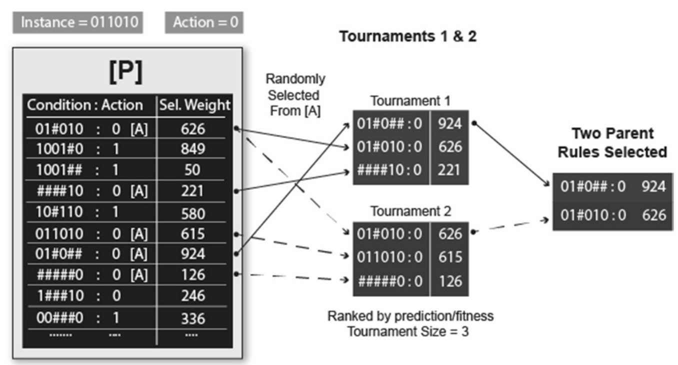
- GA rule discovery is activated if average experience of classifiers in selection pool is above a user defined cut-off.
- Classifier experience is the number of instances that the classifier has matched.

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

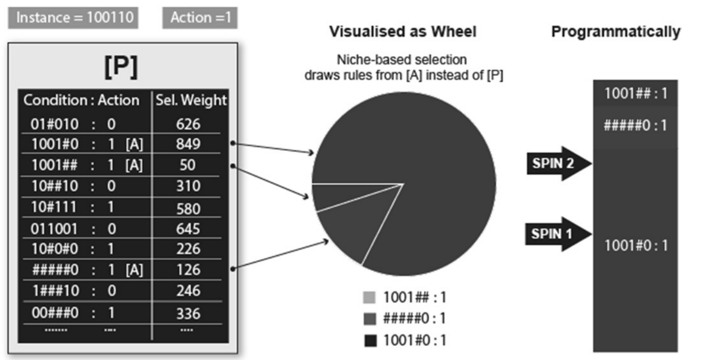
LCS Algorithm Walk-Through: Selection - Tournament



- Tournament Selection is typically used for GA parent selection

*Image adapted from [37]

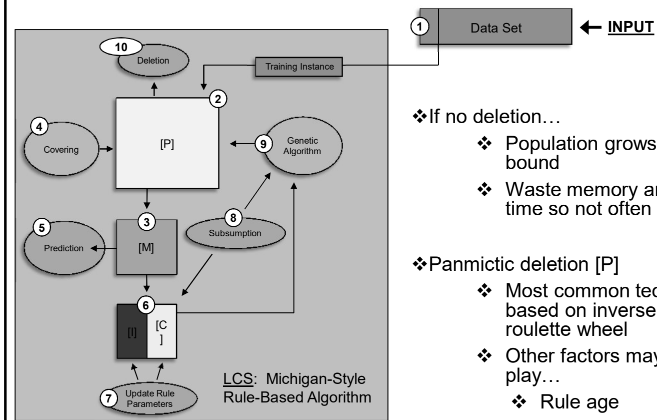
LCS Algorithm Walk-Through: Selection – Roulette Wheel



- ❖ Roulette Wheel Selection is typically used for deletion (where probability of selection is inversely proportional to fitness)

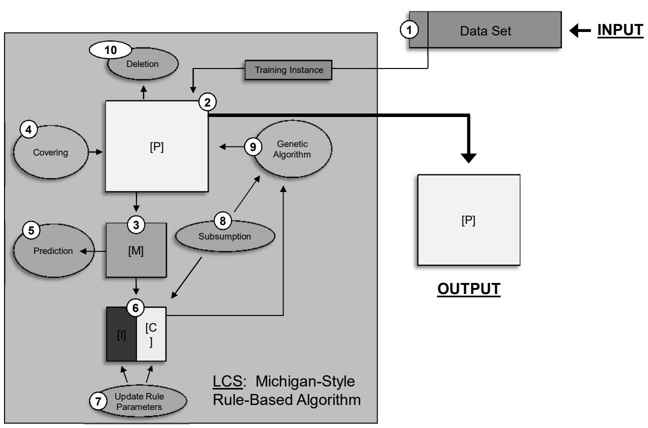
*Images adapted from [37]

LCS Algorithm Walk-Through: Deletion

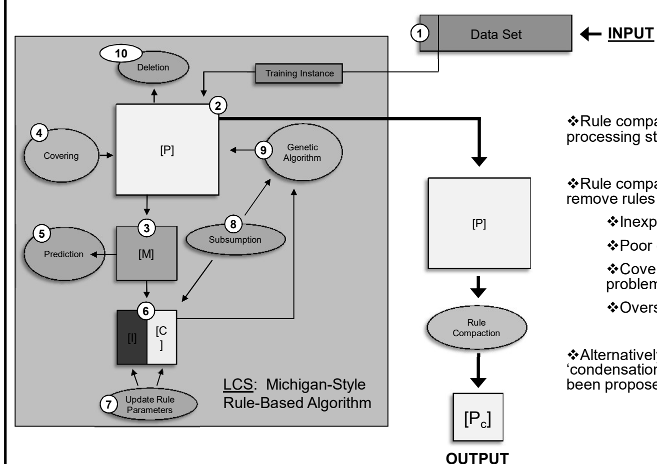


- ❖If no deletion…
  - ❖ Population grows without bound
  - ❖ Waste memory and takes time so not often used
- ❖Panmictic deletion [P]
  - ❖ Most common technique based on inverse fitness roulette wheel
  - ❖ Other factors may come into play…
    - ❖ Rule age
    - ❖ [A] size parameter
    - ❖ Numerosity

LCS Algorithm Walk-Through: Output [P]



LCS Algorithm Walk-Through: Rule Compaction



- ❖Rule compaction is a post-processing step applied to [P]
- ❖Rule compaction seeks to remove rules that are:
  - ❖Inexperienced (young)
  - ❖Poor quality
  - ❖Covering redundant problem space
  - ❖Overspecific
- ❖Alternatively similar 'condensation' approaches have been proposed to

Supervised LCS

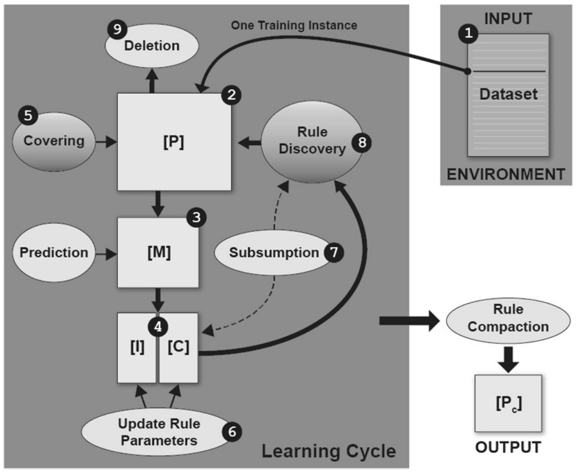


*Image adapted from [37]

Reinforcement LCS

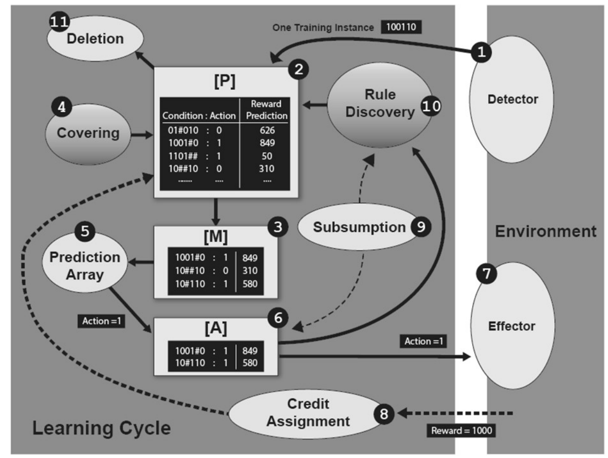
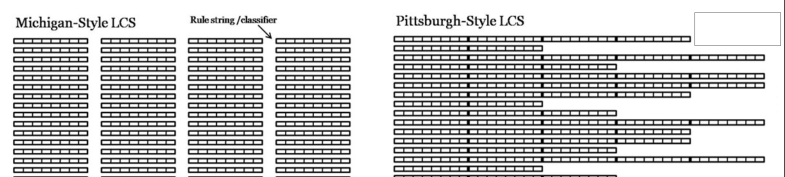


*Image adapted from [37]

Learning Classifier Systems in a Nutshell

https://www.youtube.com/watch?v=CRge_cZ2cJc

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!

Advanced Topics: Learning Parameters {1 of 2}

Parameter	Description
N	Population size
β	Learning rate for prediction, prediction error, and fitness updates
γ	Discount factor in multistep problems
θ_{GA}	Threshold for GA application in the action set
ε_0	Threshold error in prediction under which a classifier is considered to be accurate
α	Controls the degree of decline in accuracy if the classifier is inaccurate
χ	Probability of crossover per invocation of the GA
μ	Probability of mutation per allele in an offspring
ν	Fitness exponent
θ_{del}	Experience threshold for classifier deletion
δ	Fraction of mean fitness for deletion
θ_{sub}	Classifier experience threshold for subsumption
$P_{\#}$	Probability of a # at an allele position in the condition of a classifier
p_I, ε_I, and F_I	Prediction, prediction error, and fitness assigned to each classifier at the start

*Table adapted from [37]

Advanced Topics: Learning Parameters {2 of 2}

Parameter	Sym.	Initial Value	Common Range	Increment	Changeable
Environment interactions (Iterations)	I	10,000	10k - 2M	x10	Often
Population size	N	1,000	500 - 50k	± 1,000	Often
Don't care probability	$P_{\#}$	0.3	0 - 0.99	± 0.1	Often
Accuracy threshold	ε_0	0.01	0 - 0.01	±0.01	Moderately
Fitness exponent	ν	5	1 - 10	±1	Moderately
Learning rate	β	0.1	0.1-0.2	±0.02	Moderately
GA threshold	θ_{GA}	25	20-25	±5	Rarely
Mutation probability	μ	0.4	0.2-0.5	±0.1	Rarely
Crossover probability	χ	0.8	0.7-0.9	±0.1	Rarely
Classifier threshold for deletion	θ_{del}	20	20-25	±5	Rarely
Classifier threshold for subsumption	θ_{sub}	20	20-25	±	Rarely
Fitness fall-off	α	0.1	0.1	NA	Never

*Table adapted from [37]

Advanced Topics: LCS as Map Generators

- The intention is to form a *map* of the problem space
- Breaks the problem into simpler pieces as needed.

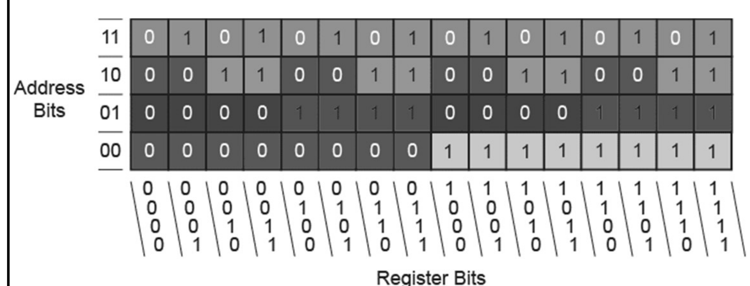


*Image adapted from [37]

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

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 (implicit ensemble)
- Each rule 'covers', i.e. describes, its part of the search space.
- The rules within a niche compete to map the domain.

Advanced Topics: Overgenerals

- 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

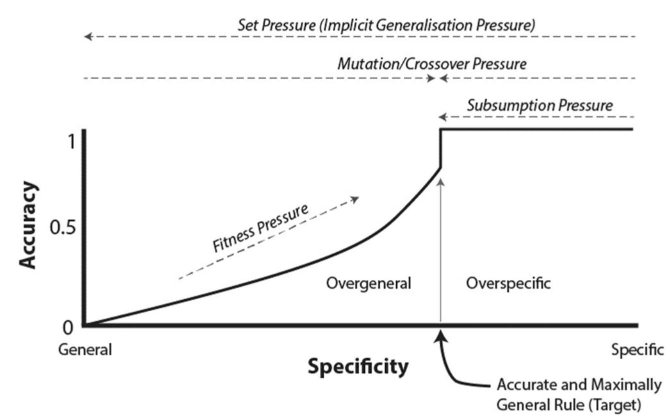


*Image adapted from [37]

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
- ❖ Favors specific rules (cover less domain)
- ❖ Fitness measures based on reward trade mistakes for more reward
- ❖ Favors 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 or 1	* Some LCS algorithms do not allow specialisation to a different state value (e.g. 0 → 1 or 1 → 0).
1 → 0 or #	1 → #	0 → 1	
# → 0 or 1		1 → 0	

strong fitness pressure

*Image adapted from [37]

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
- ❖ The danger with optimum action only is: a suboptimal rule could be converged upon … difficult to discover and switch policy. Also, no memory of bad rules is preserved.
- ❖ 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

Specific Map	Best Action Map	Complete Action Map
00 : 1 p1000	0# : 1 p1000	0# : 1 p1000
01 : 1 p1000	#0 : 1 p1000	#0 : 1 p1000
10 : 1 p1000	11 : 0 p1000	11 : 0 p1000
11 : 0 p1000		0# : 0 p0
		#0 : 0 p0
		11 : 1 p0

Boolean NAND Problem: If the two features in the condition are NOT both 1 then the class = 1, otherwise the class = 0

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]

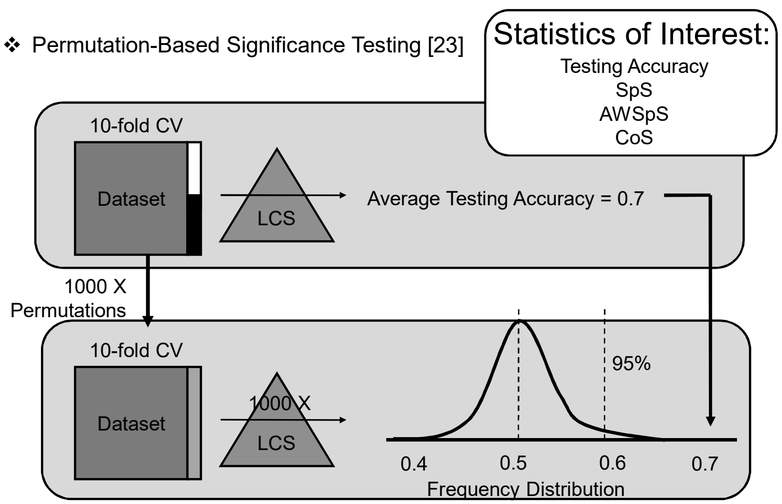
Attribute	X1	X2	X3	X4	Class	Numerosity	Accuracy
R1	X	#	#	X	0	5	0.73
R2	#	X	#	X	1	1	0.51
R3	X	X	#	X	0	2	0.88
R4	#	X	X	#	1	1	0.62
SpS	7	4	1	8			
AWSpS	5.41	2.89	0.62	5.92			

SpS (X1) = 5 + 2 = 7

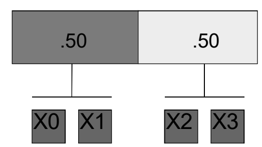
AWSpS (X1) = (0.73) * 5 + (0.88) * 2 = 5.41

Advanced Topics: Knowledge Discovery {2 of 5}

- Permutation-Based Significance Testing [23]



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]

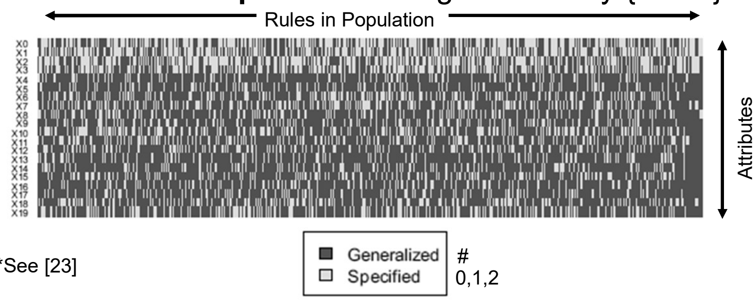
Individual Attributes

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

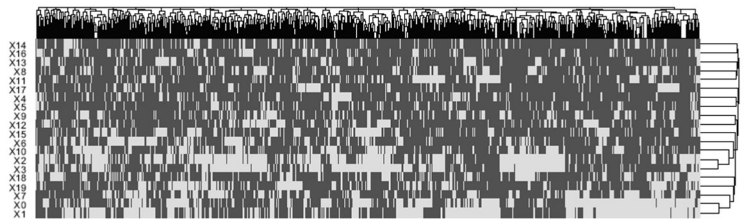
Pairs

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*

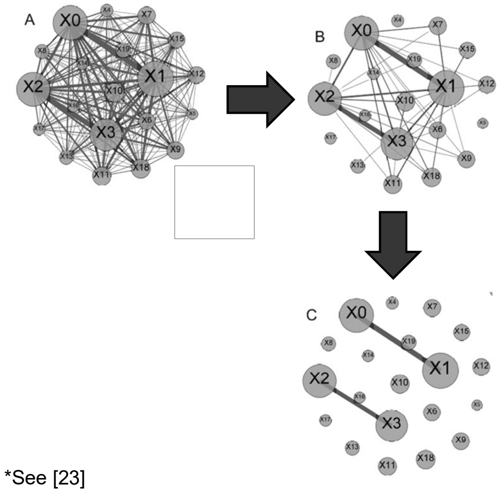
Advanced Topics: Knowledge Discovery {4 of 5}



*See [23]



Advanced Topics: Knowledge Discovery {5 of 5}



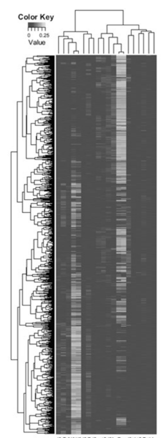
Pairs

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*

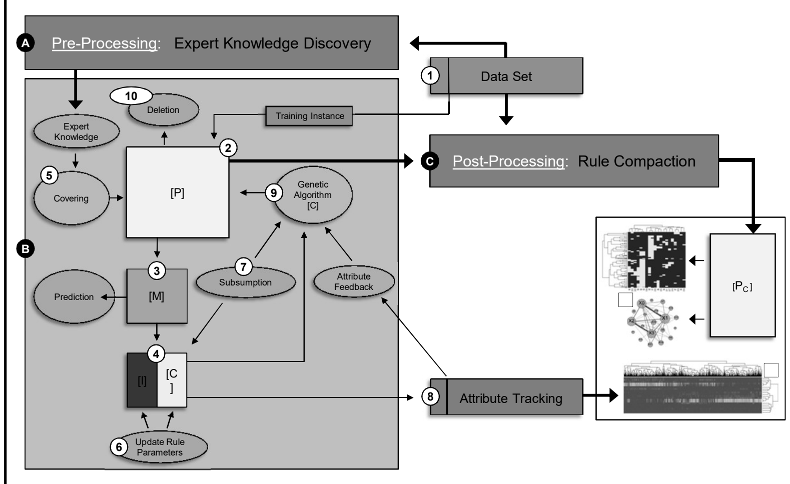
*See [23]

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.

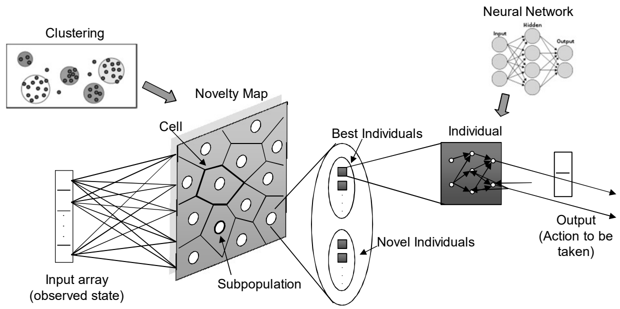


Advanced Topics: ExSTraCS A Shameless Plug



(*See [28])

- Self-Organizing Classifiers -
Automatic Adaptive Division of Input Space



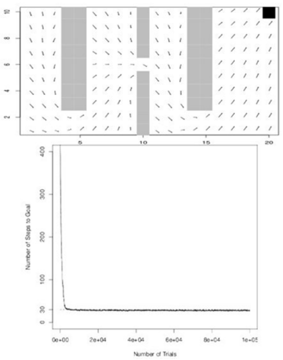
Algorithm Type	Model	Fitness
Self-Organizing Classifiers	Dynamic state-table of prediction values	Strength based
Learning Classifier Systems	Set of condition-action-prediction rules	Accuracy based
Standard Reinforcement Learning	Static state-action lookup tables	Strength based

(*See [40])

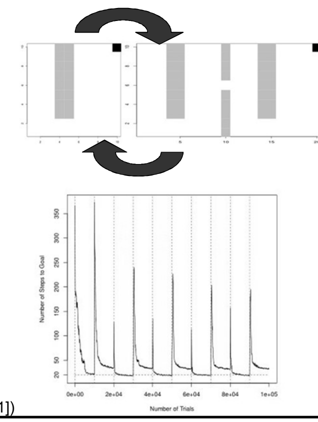
Results: Maze

One of the few methods that solve noisy mazes and the only one capable of solving dynamic mazes.

Noisy



Dynamic

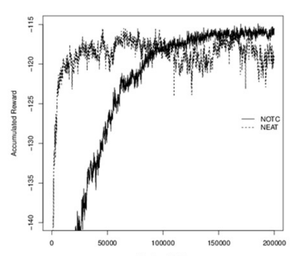


(*See [41])

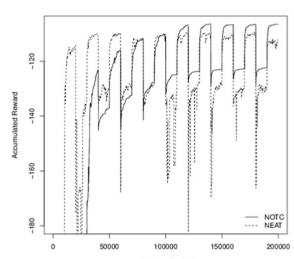
Results: Mountain Car

Illustration of Mountain Car

Noisy



Dynamic



(*See [42])

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

Resources – Software

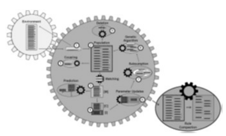
- Educational LCS (eLCS) – in Python.
  - https://github.com/ryanurbs/eLCS
  - Simple Michigan-style LCS for learning how they work and how they are implemented.
  - Code intended to be paired with first LCS introductory textbook by Urbanowicz/Browne.
- ExSTraCS 2.0 – Extended Supervised Learning LCS – in Python
  - https://github.com/ryanurbs/ExSTraCS_2.0
  - 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:
  - 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.

New Resources

- Textbook: Introduction to Learning Classifier Systems (Urbanowicz & Brown, 2017). Now available from Springer.
- YouTube video on LCS:
  - Learning Classifier Systems in a Nutshell
  - Animated, narrated explanation of basic LCS concepts.
  - https://www.youtube.com/watch?v=CRge_cZ2cJc



- LCS and Rule-Based Machine Learning Wikipedia Pages – recently updated and revised. (https://en.wikipedia.org/wiki/Learning_classifier_system)
- Please join us for the International Workshop on Learning Classifier Systems (IWLCS)

Conclusions

- 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
- Increasing 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.
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.

References {2 of 4}

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.
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.

References {3 of 4}

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.
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.
37) Urbanowicz, Ryan J., and Will Browne. "An Introduction to Learning Classifier Systems". Springer, 2017
38) Forrest, S., & Jones, T. (1994). Modeling complex adaptive systems with Echo. Complex systems: Mechanisms of adaptation, 3-21.
39) Holland, J. H. (1992). Complex adaptive systems. Daedalus, 17-30.

References {4 of 4}

40) Vargas, D. V., Takano, H., & Murata, J. (2013). Self organizing classifiers and niched fitness. In Proceedings of the 15th annual conference on Genetic and evolutionary computation (pp. 1109-1116). ACM.
41) Vargas, D. V., Takano, H., & Murata, J. (2013). Self organizing classifiers: first steps in structured evolutionary machine learning. Evolutionary Intelligence, 6(2), 57-72.
42) Vargas, D. V., Takano, H., & Murata, J. (2015). Novelty-organizing team of classifiers in noisy and dynamic environments. In Evolutionary Computation (CEC), 2015 IEEE Congress on (pp. 2937-2944). IEEE.
43) Wolfram, S. (1986). Theory and applications of cellular automata: including selected papers 1983-1986. World scientific.