Chapter 1

EC-Star: A MASSIVE-SCALE, HUB AND SPOKE, DISTRIBUTED GENETIC PROGRAMMING SYSTEM

Una-May O'Reilly[1], Mark Wagy[1] and Babak Hodjat[2]
[1]*Evolutionary Design and Optimization Group, CSAIL, MIT,* [2]*Genetic Finance, CA 94105 USA.*

Abstract

We describe a new Genetic Programming system named EC-Star. It is supported by an open infrastructure, commercial-volunteer-client, parallelization framework. The framework enables robust and massive-scale evolution and motivates the hub and spoke network topology of EC-Star's distributed GP model. In this model an Evolution Coordinator occupies the hub and an Evolutionary Engine occupies each spoke. The Evolution Coordinator uses a layered framework to dispatch high performing, partially evaluated candidate solutions for additional fitness-case exposure, genetic mixing and evolution to its Evolutionary Engines. It operates asynchronously with each Evolutionary Engine and never blocks waiting for results from an Evolutionary Engine.

Keywords: genetic programming, cloud-scale, distributed, learning classifier system

1. Introduction

In this chapter we introduce a platform named EC-Star. The platform pioneers a distributed Genetic Programming (GP) model upon a commercial volunteer[1] resource, parallel computing framework. It is elastic: volunteer nodes can independently enter the framework and the evolutionary algorithm can seamlessly integrate them as Evolutionary Engines which each independently interact with the Evolution Coordinator. Evolutionary Engines can withdraw from volunteering and the evolutionary computation and Evolution Coordinator continue seamlessly without them. EC-Star could feasibly be used to solve problems that have to date been considered intractable because it is able to integrate a virtually limitless number of "come-and-go", volunteer, compute nodes.

EC-Star's distributed GP model is arranged in a unique hub and spoke topology. In this topology an Evolution Coordinator occupies the hub and an Evolutionary Engine occupies each spoke. The Evolution Coordinator dispatches high performing, partially evaluated candidate solutions for additional fitness evaluation, genetic mixing and evolution to its Evolutionary Engines. It exploits *archive layering* as a means of ensuring genetic diversity and fostering open-ended evolution. EC-Star's representation for a candidate solution is a classifier composed of multiple rules. EC-Star is able to cope with the complexity and extensive evolution required by this representation because of its massive scale.

To date, the EC-Star has been successfully used on financial time series data to trigger "buy", "sell" and "hold" trading signals for stock portfolios. It has significant potential to be used for other time series applications, such as predicting trends in biological signals as well as standard supervised (e.g. classification) and unsupervised (e.g. clustering) machine learning approaches on large datasets.

We proceed as follows: In Section 2 we describe EC-Star's architecture. In Section 3 we describe EC-Star's classifier representation and how a classifier is evaluated on a fitness case. In Section 4 we discuss some of EC-Star's novel features and compare it to other distributed GP models. Section 5 summarizes.

2. EC-Star Architecture

EC-Star has two software layers; a parallelization framework lower layer which supports a distributed GP model upper layer, as shown in

[1]The compute resource is commerical-volunteer in the sense that it is doing work on behalf of someone else, but is being paid to do it

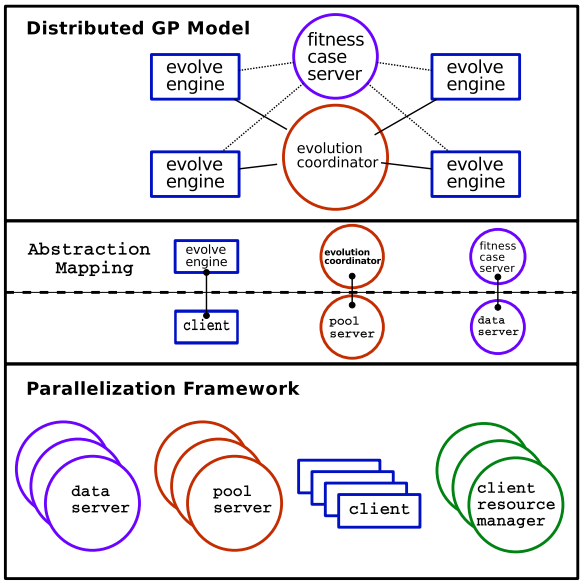


Figure 1-1. EC-Star system framework architecture. The lower layer is a parallelization framework. Abstracted above it is a distributed GP model.

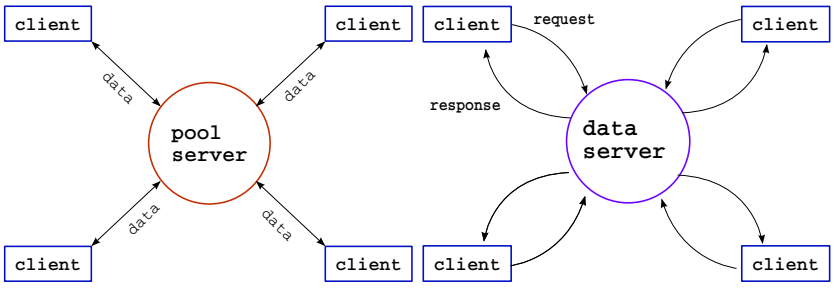
Figure 1-1. The lower layer is an open-infrastructure, volunteer compute framework, similar to that of BOINC (Anderson, 2004) (though different in the sense that it is a *commercial* volunteer network) which is perhaps familiar to readers as the framework supporting the SETI@Home project (Anderson et al., 2002). This layer enables the upper layer's distribution and aggregation of evolutionary genetic material and distributed algorithmic control. The upper layer is a unique hub and spoke distribution model of asynchronous evolutionary computation.

Parallelization Framework Layer

The framework has dedicated resources termed *servers* and commercial volunteer compute nodes termed *clients*. Clients are volunteers in the sense that they offer to compute in their "spare time" using idle cycles that their primary applications leave unused. Given this scheme, a client does not deliver any work on a deadline. Nor does it, in fact, commit to ever completing its work. It computes on behalf of the framework by means of executing a program received from EC-Star. It executes the "guest" program on behalf of EC-Star usually in its background. Clients do not communicate with one another, thus assuring their privacy. A client is persistent in the sense that it can store a state file on its disk

and shut down its program. Then when it decides to volunteer again, it can resurrect itself using the state file to resume where it left off.

The framework's dedicated resources and volunteer compute resources can communicate. This encourages the former to function as hubs while clients function as spokes. The framework designates specialized dedicated resources termed Pool Servers, Client Resource Managers and Data Servers. The Pool Server (Figure 1-2a) functions as a center of communication with all clients. It globally coordinates clients after they have volunteered, similar to the master server in the BOINC system (Anderson, 2004), which coordinates the activities of the volunteer compute nodes. The Data Server (Figure 1-2b) serves data requested by clients. The *Client Resource Manager* is a container for groups of clients. It launches and shuts down clients; and it distributes client state amongst its grouping of clients if they are inactive or down, thus achieving some level of *Failure Transparency* in the system to client inactivity.



(a) Pool Server (b) Data Server

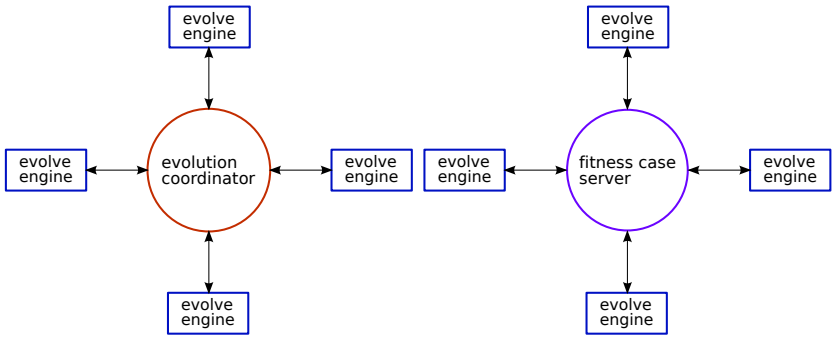
Figure 1-2. The Pool Server coordinates client activity once a client volunteers.Clients request Data Packages from the Data Server, and the Data Server sends a random Data Package in response.

The parallelization framework is resilient because the servers can continue to function as clients volunteer and retire completely at their own discretion. This is because clients initiate their startup as well as any other communication with the Pool Server and Data Server. The Data Server only computes in response to a pull from a client. The Pool Server computes asynchronously, in a non-blocking manner, with clients: it can proceed without ever blocking to await a client's message or response. This resiliency and the Client Resource Manager are completely hidden from the upper layer.

Distributed GP Model Layer

The natural hub and spoke relation between a server and clients in EC-Star's parallelization framework is exploited directly by EC-Star's hub and spoke distributed GP model topology. An Evolution Engine is the program that EC-Star runs on a client. The Evolution Coordinator is a hub that runs on the Pool Server, see Figure 1.3(a). The data that is routed to and from the Evolution Coordinator to the Evolution Engine is a set of individuals also known as classifiers. The Data Server runs a Fitness Case Server, which distributes fitness cases to each Evolution Engine, see Figure 1.3(b)). The Client Resource Manager does not have a corresponding element in the distributed model – it is completely hidden from the distributed layer.

At first glance, the hub and spoke GP model in EC-Star appears to be a special topological case of the coarse-grained, island model GP. However, the key difference is that, in the case of the island model, each island sends a set of high fitness individuals to one or more others until some sort of halting condition is reached. In EC-Star, there is no communication between the clients. In this respect, EC-Star's model in an abstract way, supports a dynamic topology because the Evolution Coordinator collects genetic material from all Evolution Engines and directs it to any other.



(a) Evolution Coordinator (b) Fitness Case Server

Figure 1-3. The Hub and Spoke distributed model. The Evolution Coordinator coordinates genetic matierial. The Fitness Case Server provides fitness cases to each Evolve Engine.

EC-Star's Evolution Coordinator

The Evolution Coordinator maintains a layered *Archive* – a sorted set of individuals that are currently the best from all Evolution Engines. The top level of this Archive consists of the best individuals at a given time – i.e. the solution. The Evolution Coordinator uses the Archive as a reservoir of individuals to be sent out to each Evolution Engine in order to improve fitness and possibly mix their genetic material with the endemic population that evolves on each Evolution Engine. Individuals are stored in the Evolution Coordinator Archive in layers. The number of layers and individuals per layer is fixed. When a new or returning individual comes to the Evolution Coordinator, first its updated fitness is reconciled with any other fitness updates that may have been reported by other Evolution Engines that also evolved it (see section 4.0 for more details). The updated individual then competes with a layer of similar individuals for a slot if that layer is already full.

EC-Star's Evolution Engine

Each Evolution Engine is an execution environment of evolutionary computation. It has a defined target population size. At initialization, see Algorithm 1, it receives a set of features (indicators) and some individuals from the Evolution Coordinator before generating the rest randomly. To execute its evolutionary loop, it obtains fitness cases (training data from a machine learning perspective) from the Fitness Case Server as a *Data Package*. Each classifier of a population is evaluated on the Data Package, and after a fixed number of Data Packages – the *Training Window* – selection, reproduction and variation take place and the next generation's population is again tested and evolved. It periodically receives individuals from the Evolution Coordinator to integrate into its current population. To the Evolution Coordinator, it reports its best individuals and returns updated fitness information on the individuals it receives. See Algorithm 2 for more details.

Algorithm 1 Evolution Engine: Initialization

$coordinateSoftwareVersionWithEvolutionEngine()$
$featureSet \leftarrow EvolutionCoordinator.getFeatureSet()$
$population \leftarrow EvolutionCoordinator.fetchIndividuals()$
if $population.size < TARGET\_SIZE$ **then**
  $population += generateRandomIndividuals()$
end if

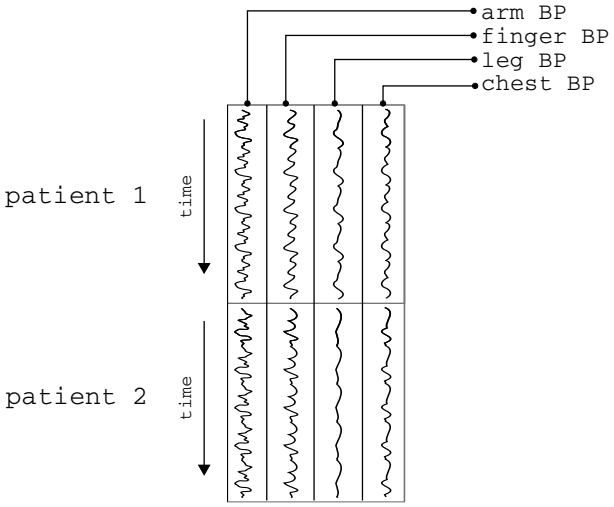


Figure 1-4. A Data Package example illustrating the layout of blood pressure data collected at different parts of the body for three patients.

The Fitness Case Server

The Fitness Case Server disseminates random packages of fitness cases upon Evolution Engine requests. A Data Package consists of column-wise features and row-wise time-series data. Rows of time-series data can additionally be grouped into *Bins*. For example, given measurements of blood pressure at different locations of the body in multiple patients, each column in a Data Package is a measurement time at a different body location, each row is be a set of measurements recorded at the same time, and one Bin – set of rows – is a different patient's time-series values (see Figure 1-4).

Each generation, the population on an Evolution Engine is evaluated with respect to a certain predefined number of Data Packages. This quantity is only a subset of the entire data set implying that evolution takes place while the population is progressively evaluated against more Data Packages, see Section 3.0 and Chapter **??** for more details on this "age-varying fitness estimation".

3. EC-Star's Classifier Representation and Fitness Evaluation

In EC-Star, an individual or *classifier* consists of an age, set of rules and a fitness, see Table 1-1. Each rule is a size-varying conjunctive set

Algorithm 2 Evolution Engine: Evolution

```
1: loop
2:     for all dataPackageIdx = 1 to TRAINING_WINDOW_SIZE do
3:         dataPackage ← dataServer.getDataPackage()
4:         for all bins ∈ dataPackage.bins do
5:             for all individuals ∈ population do
6:                 matchSet ← []
7:                 for all dataPoints ∈ bin do
8:                     for all rules ∈ ruleset do
9:                         if allrule.conditions == True then
10:                            matchSet.add(rule)
11:                        end if
12:                    end for
13:                    chosenRule ← pickSingleRule(matchSet)
14:                    useAction(chosenRule)
15:                end for
16:                individual.age + +
17:            end for
18:        end for
19:    end for
20:    reportPopulationInfoToEvolutionCoordinator(population)
21:    newPopulation ← EvolutionCoordinator.fetchIndividuals()
22:    elitistPool ← getBest(population)
23:    newPopulation+ = evolve(elitistPool)
24:    if newPopulation.size < TARGET_SIZE then
25:        newPopulation+ = generateRandomIndividuals()
26:    end if
27:    classId ← EvolutionCoordinator.checkClassId()
28:    savePopulationStateToDisk()
29:    population ← newPopulation
30: end loop
```

$< classifier >$	::=	$< age >< fitness >< rules >$
$< rules >$	::=	$< rule > \mid < rule >< rules >$
$< rule >$	::=	$< conditions >< action >$
$< conditions >$	::=	$< condition > \mid < condition > \wedge < conditions >$
$< action >$	::=	prediction label
$< condition >$	::=	$< predicate > \mid \neg < condition > \mid < condition > [lag]$
$< predicate >$	::=	truth value on a feature indicator

Table 1-1. Classifier definition in BNF notation. The lag operator refers to a past instance of a value to be used in a given condition.

of conditions with an associated action, the latter of which represents a class in a classification problem . Each condition acts as a propositional variable, which is then applied to the real-valued training environment. Apart from conjunction operators, a *complement* operator and a *time-lag* operator can be applied to each condition. The complement operator negates the condition's truth value, whereas the time-lag operator refers to "past" values of an attribute. See Figure 1-5 for an example of an individual.

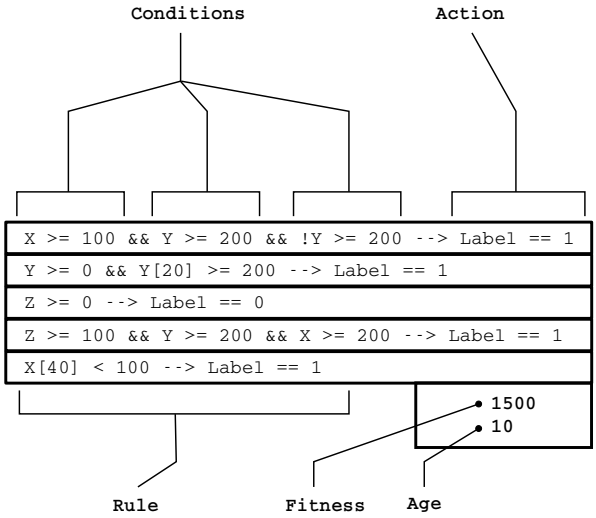


Figure 1-5. An example of what an individual or classifier looks like.

The representation of individuals in EC-Star is very similar to a classifier in the so-called *Pitt-Style* version of a *Learning Classifier System* (LCS)(Urbanowicz and Moore, 2009), (Jong et al., 1993). As in a Pitt-style LCS, the individual or classifier represents a full solution-space –

that is, each individual contains the rules needed to classify a row of test data (in distinction to the Michigan-style of LCS in which all classifiers together represent a classification solution). And like Pitt-style LCS, each rule-set is assigned a fitness.

Fitness Evaluation of a Classifier

Each individual in an Evolution Coordinator's population is evaluated with respect to a set of fitness cases (a.k.a training data) which spans one or more Data Packages. For each fitness case (or row of data), the variables in the classifier's rules' conditions are bound to the features of the fitness case. For each rule of the classifier, the body of the rule – that is, the rule's conditions – is tested for its truth value. If all conditions in the rule's body evaluate to true, the rule is added to a *Match Set*. When all rule bodies have been tested, a voting mechanism referencing the Match Set elects a single rule's action to "act" on behalf of the classifier. This action becomes the classifier's predicted class for the fitness case. The prediction is compared against the actual class (available in a special column) and fitness is assigned according to whether there is agreement. Fitness scoring takes place for each row of fitness case data in a Data Package and a set number of Data Packages before a new population is created through selection, mutation and crossover. See Algorithm 2 for pseudocode of this process.

Genetic Variation

In EC-Star conditions are the most primitive evolvable genetic unit. Evolutionary variation can also take place within rule sets via condition and action crossover and mutation. It can also take place across rules by rule exchange or addition. When crossover takes place between rules, actions are swapped.

EC-Star imposes certain restrictions on the variation and creation of rules to avoid tautologies and logical fallacies. This includes requiring that no two conditions of a rule can reference the same feature, unless they have different time lags. This restriction helps to control rule *bloat*.

4. Other Aspects of EC-Star

Apart from its novel distributed GP model supported by a robust commercial volunteer-compute framework, EC-Star has other aspects worthy of description and related to previous work.

Experience Archive Layering Population Structure

One particularly interesting feature of the EC-Star is its Experience Archive Layering Population Structure, "Experience Layering", at Evolution Engines and the Evolution Coordinator. This archival layering of classifiers is similar and inspired by, but not precisely the same as, Age-Layered Population Structure (ALPS) (Hornby, 2006) or Hierarchical Fair Competition (HFC) (Hu and Goodman, 2002).

In HFC, individuals are put into layers, but the layers are based on their fitness. In Experience Layering, individuals are layered by their experience on the training data. This is very similar to ALPS, with the distinction of how an individual's "age" is defined. In ALPS, age is defined as the number of generations of evolution that an individual has survived. However in Experience Layering, the idea of age is one of how many training data points an individual has seen up to that point.

Experience Layering offers diversity promotion: it is promoting individuals of diverse experience. It encourages diversity because if an individual with experience just over the lower threshold of a given layer is inserted into layer L_i, another individual with an experience level just below the lower threshold of layer L_i competes for a slot in layer L_{i-1} and it is shielded from competing with the former individual, despite both having very similar experience.

Another purpose of Experience Layering is to serve the Evolution Coordinator as a resource for selecting individuals to send to Evolution Engines for improved fitness estimation. At each Evolution Engine, an individual will be evaluated against new fitness cases. This contrasts with ALPS isolated selection process per layer.

Evolvable Time Lags

Another notable aspect of the EC-Star is a lag, which is optionally appended to a condition (see Table 1-1). A lag supports efficient representation of time-series data in the Data Package. Instead of requiring a single time-series and delayed copies of the same time-series as separate fitness cases, the lag allows a more compact time series representation which is reference when fitness is tested. Only one copy of the time-series needs be included in a Data packer, and the fitness test procedure uses a pointer to "past" events matched to the lag.

In the context of time-series prediction, the evolvability of this lag is particularly interesting. The fitness of an individual is computed with respect to values that have occurred in "the past" relative to the current fitness case but the precise lag does not have to be identified ahead of time. Evolution can find the appropriate lag.

The ability to evolve the lag and calculate fitness with lags built into conditions allows reward to be allocated to an individual with respect to the events in time that occurred in the past rather than just at a single point in time. This serves a similar purpose to the mechanism of delayed reward propagation in reinforcement learning (Goldberg, 1989),(Wilson, 1995) but without the need to explicitly maintain a data structure for classifier memory.

Rule Election

When an individual is evaluated against a fitness case, all of the individual's rules that evaluate to "true" are collected into a Match Set. An election strategy must be implemented to choose between equally valid rules in the Match Set. EC-Star supports any election strategy. For example, one strategy could pick a rule at random from the Match Set. Another can elect the rule that has fired most frequently over the course of the classifier's lifetime. The strategy has significant implications in the evolved capability of EC-Star. Design and comparison of different election strategies will be a topic of future work.

Feature Selector

EC-Star assumes very high dimensional data. Thus it maintains a Feature Selector: a repository of feature subsets of the full feature space. This Feature Selector creates new subsets of features which it initially sends out in response to an Evolution Engine's request when the engine is initializing. It then monitors the progress of each feature subset as Evolution Engines report their fitness progress to the Evolution Coordinator. Depending on the performance of each subset of features, the Feature Selector potentially removes sets or merges them in an effort to direct evolution with the most strongly indicative feature set.

5. Summary

EC-Star is characterized by its:

- massive distribution capacity derived from being able to enlist come-and-go commercial volunteer compute nodes while running multiple, non-blocking, dedicated resources.
- capacity for large numbers of fitness cases and cost-effectie means of sampling them using "age-varying" fitness estimation (see Chapter **??** by Hodjat and Shahrzad).
- hub and spoke topology for distributed GP.
- Experience Archive Layering Population Structure, Experience Layering. This layering of individuals according to MasterFitness fosters a diverse population and encourages open ended evolution.
- elasticity: EC-Star computes for an open ended duration during which its resource capacity, in terms of Evolution Engines, can expand and contract.
- scalability: The Pool Server/Evolution Coordinator is capable of handling many clients and multiple Pool Servers can be deployed to extend the amount of clients available for computation. Multiple Data Servers/Fitness Case Servers can be deployed to scale up to larger and larger amounts of data to be used as fitness cases. Multiple clients/Evolution Engines can come and go as volunteers.
- classifier representation where a classifier is composed of multiple rules, each with variable numbers of conditions. EC-Star supports different election strategies for matched rules election.
- robustness: Clients/Evolution Engines can go up and down without negatively impacting the overall system. A Pool Server/Evolution Coordinator is a single point of failure but fairly lightweight. When a failed Pool Server/Evolution Coordinator comes back online, clients/Evolution Engines will continue to update it with individuals which they have been evolving.
- Evolvable Time Lags: Lags support a compact representation of fitness case data and imply the references to past events in time-series data do not have to be pre-determined.
- Feature Selector: Subsets of the feature space are tracked for convergence. This exposes those features most germane to guiding the solution toward convergence.

Acknowledgments

The authors acknowledge the generous support of the Li Ka Shing Foundation as well as Kaivan Kamali and Hormoz Shahrzad of Genetic Finance and Kalyan Veeramachaneni of MIT.

References

Anderson, David P., Cobb, Jeff, Korpela, Eric, Lebofsky, Matt, and Werthimer, Dan (2002). Seti@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61.

Anderson, D.P. (2004). BOINC: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 4 – 10.

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass.

Hornby, Gregory S. (2006). ALPS: the age-layered population structure for reducing the problem of premature convergence. In Keijzer, Maarten et al., editors, *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, volume 1, pages 815–822, Seattle, Washington, USA. ACM Press.

Hu, Jianjun and Goodman, Erik D. (2002). The hierarchical fair competition (HFC) model for parallel evolutionary algorithms. In Fogel, David B. et al., editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 49–54. IEEE Press.

Jong, Kenneth A. De, Spears, William M., and Gordon, Diana F. (1993). Using genetic algorithms for concept learning. *Machine Learning*, 13.

Urbanowicz, Ryan J. and Moore, Jason H. (2009). Learning classifier systems: A complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009. Article ID 736398.

Wilson, Stewart W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175.