XCSJava 1.0: An implementation of the XCS classifier system in Java

Martin V. Butz

IlliGAL Report No. 2000027
June, 2000

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue Urbana, IL 61801
Office: (217) 333-2346
Fax: (217) 244-5705

XCSJava 1.0: An implementation of the XCS classifier system in Java

Martin V. Butz*
Illinois Genetic Algorithms Laboratory
Department of General Engineering
University of Illinois at Urbana-Champaign
butz@illigal.ge.uiuc.edu

Abstract

The XCSJava 1.0 implementation of the XCS classifier system in Java is freely available from the IlliGAL anonymous ftp-site. The implementation covers the basic features of the XCS classifier system and provides a multiplexer and maze environment for testing purposes. This paper explains how to download, compile, and run the code. Moreover, it explains the object oriented approach in the implementation and the possible parameter manipulation as well as the environmental interface to hook in other test environments. Additionally to the source code, an executable package of the version as well as an XCSJava 1.0 API documentation is provided.

1 Introduction

This paper serves as a manual for using the XCSJava 1.0 implementation. The code as well the Java byte code is available form the IlliGAL anonymous FTP-site. Due to the platform independence of Java the code should be executable on any system with an appropriately set up Java environment. Note, that this is not a Java applet but a Java application. Thus, it is not executable in a Browser.

The code realizes the XCS classifier system as explained in Wilson (1995) and enables subsumption (Wilson, 1998). Furthermore, the modifications in Wilson (1999) are considered except for the enhancement to real valued coding. The code was tested on different Woods environments with different codings as well as the Multiplexer problem. Essentially, the code stays as close as possible to the recently published algorithmic description of the XCS classifier system (Butz & Wilson, 2000).

At first, the paper gives instructions how to download the code and extract the package including the source code, an executable jar-file of the code, different coded mazes, and the XCSJava 1.0 API documentation. Next, it describes how to compile and run the code. Section 3 describes the different objects and their interaction. It is also explained, how to select different problems and how to implement new problems in this framework. Next, Section 4 reveals how the parameters can be manipulated and the different features can be selected. Finally, Section 5 describes how the output is structured.

*Visiting student from the University of Würzburg, Institute for Psychology III, Germany

2 How to Download, Extract, Compile, and Run the code

The package with the source code and some examples is available at the IlliGAL Anonymous FTP site in ftp://ftp-illigal.ge.uiuc.edu/pub/src/XCSJava/XCSJava1.0.tar.Z.

2.1 Download and Extract

After downloading the package (XCSJava1.0.tar.Z) into a directory, the directories and files can be extracted by typing:

```
uncompress XCSJava1.0.tar.Z
tar xvf XCSJava1.0.tar
```

When extraction the files a new directory, called `XCSJava`, will be created that contains the following:

```
API                 MazeEnvironment.java  XCSJava1.0.jar
Environment.java    PredictionArray.java  XClassifier.java
Environments        XCS.java              XClassifierSet.java
MPEnvironment.java  XCSConstants.java
```

In the `Environments` subdirectory the maze environments Woods1, Woods2, Maze4, Maze5, and Maze6 are provided. The XCSJava 1.0 API can be found in the folder `API`. The executable application is the file `XCSJava1.0.jar`.

2.2 Compile

The compilation of the source code is executed in the `XCSJava` directory with the command `javac *.java`.
However, for first testing purposes the executable XCSJava1.0.jar file is provided which is executable on any system with an appropriately set up Java runtime environment.

2.3 Run the Code

The provided `XCSJava1.0.jar` application can be executed by typing:

```
java -classpath XCSJava1.0.jar XCS
```

If the code is recompiled (e.g. for changing certain parameter settings), the created class files can be executed by typing:

```
java XCS
```

In some Java runtime environments the path of the Java package `classes.zip` needs to be specified as well in the `-classpath` option (separated by a semicolon). The package is usually found in the `jdkX.X.X/lib/` folder.

The XCSJava implementation asks for at least four arguments when executed. (1) The problem which it is supposed to solve. Right now, this can either be `mp` for the multiplexer problem or `maze` for a maze problem. (2) The file where the XCSJava program will write its performance. (3) If the multiplexer problem is chosen, this argument specifies the problem length of the multiplexer

problem. Note that in this implementation any length can be chosen. The program creates the largest multiplexer problem that fits in the specified length and assigns the additional bits randomly without relevance (e.g. a length of 13 will create a 11 multiplexer problem with two additional irrelevant random bits). If the maze problem is chosen, this argument specifies the file name where the maze is coded that the XCS is supposed to learn. Examples of such mazes are given in the folder `Environments`. (4) If the multiplexer problem is chosen, this argument specifies if either a 1000/0 payoff is provided (argument=0) or a payoff map is provided (argument=1) similar as described in Wilson (1995). If the maze environment is chosen, this argument specifies if each perceived position is coded with two or three bits (e.g. Woods2 is usually coded with three bits, while Maze4 is coded with two bits).

Moreover, optional another one or two arguments can be provided. (5) This argument specifies the number of explore trials which are solved in each experiment. The default is set to 20000 exploration trials. (6) Here, the number of experiments that will be executed are specified. The default is set to ten experiments.

When specifying a wrong number of arguments, the program provides a reminder of the necessary arguments. A typical call of the 11-multiplexer problem would be:

```
java -classpath XCSJava1.0.jar XCS mp XCSJava_MP11.txt 11 0
```

A typical call of the Woods2 problem would be:

```
java -classpath XCSJava1.0.jar XCS maze XCSJava_Woods2.txt Environments/Woods2.txt
3 5000
```

3 The XCSJava Code

This section summarizes the different source code files of the XCSJava code. It outlines how the XCS classifier system is divided into different objects. Also, it outlines how to plug-in other test environments into the XCSJava program as described in section 3.6. A more precise description of the functions and parameters in each class can be found in the XCSJava API documentation.

3.1 The XCS Class

The `XCS.java` file contains the XCS class and the main() function. The XCS class specifies the main learning loop in the XCS classifier system. It stores the current population of classifiers as well as the problem that is investigated by the XCS. The class distinguishes between single- and multi-step environments. Moreover, it handles the performance output.

3.2 The XClassifierSet Class

The class implemented in `XClassifierSet.java` handles a set of classifiers. It provides constructors for generating empty populations, match-sets and action-sets. Moreover, it handles the parameter update in a set, the GA application in a set, subsumption, and deletion.

3.3 The XClassifier Class

This class handles one classifier object. Constructors for generating random classifiers, randomly matching classifiers, randomly matching classifiers with specified action, and copies of specified classifiers are implemented. It provides the parameters of a classifier, as well as the different parameter updates. Moreover, it determines if the classifier is equal or more general than another classifier as well as if the classifier is a possible subsumer. Finally, it provides two-point crossover and mutation methods.

3.4 The XCSConstants Class

As further described in section 4 this class provides all the necessary learning parameters and additional flags for XCS. Each above specified class keeps one static instance of the XCSConstants class in order to access the parameters efficiently.

3.5 The PredictionArray Class

In this class all operations with a prediction array are handled. It provides a constructor for generating a prediction array out of a given set. Moreover, it handles the different types of action selection.

3.6 The Environment Interface

This interface must be implemented by all problems that are posed to the XCSJava program. Once all the functions specified in this interface are properly implemented by a class, the class can be easily hooked into the XCSJava program as a new problem. What remains to do is to construct the environment in the main() function properly and then construct the XCS object with this new environment.

3.7 The MPEnvironment Class

This class handles the multiplexer problem. As described above, it implements the Environment interface and is constructed in the main function. This implementation of the multiplexer problem provides either a payoff of 1000/0 or a reward map as specified in Wilson (1995). Moreover, the problem length is not restricted to the defined multiplexer problems but can be any integer number. The exceeding bits are generated randomly and are irrelevant for the problem.

3.8 The MazeEnvironment Class

The MazeEnvironment provides a class that can handle different mazes. The constructor requires a file in which a proper maze must be coded. Examples of coded mazes can be found in the `Environments` directory. It enables the movement in the maze and codes the perceptions in the current position. Once a food position is reached, the environment provides a reward of 1000 and requires a reset call.

4 Parameter Settings

This section describes how to modify the parameters in the XCSJava program.

The parameters are defined and can be modified in the `XCSConstants.java` file. Moreover, the file provides some other constants which support the selection of certain methods in the XCS mechanism. The following parameters are provided:

- **maxPopSize** specifies the maximal number of micro-classifiers in the population.
- **alpha** is the rate of distinction between accurate and non-accurate classifiers.
- **beta** is the learning rate.
- **gamma** is the discount factor in multi-step problems.
- **delta** specifies the fraction of the mean fitness in the population under which the fitness of a single classifier is considered in the deletion method.
- **nu** used in the power function in the fitness evaluation of a classifier.
- **theta_GA** specifies the GA threshold.
- **epsilon_0** is the error threshold under which the accuracy of a classifier is set to one. Similar to the algorithmic description this constant must be set according to the maximal payoff possible in an environment (usually to one percent of this number).
- **theta_del** is the experience threshold over which the fitness of a classifier may be considered in the deletion method.
- **pX** specifies the probability of applying crossover during a GA application.
- **pM** specifies the probability of mutating one attribute in the condition or the action of a classifier during a GA application.
- **P_dontcare** is the probability of using a '#'-symbol in one attribute during covering.
- **predictionErrorReduction** is the prediction error reduction when creating a new classifier.
- **fitnessReduction** is the fitness reduction when creating a new classifier.
- **theta_sub** specifies the required experience of a classifier to be able to subsume other classifiers.
- **teletransportation** is the maximal number of steps executed in one trial in a multi-step problem.
- **doGASubsumption** is a flag if GA subsumption should be executed.
- **doActionSetSubsumption** is a flag if action set subsumption should be executed.
- **predictionIni** specifies the initial prediction value of a new classifier.
- **predictionErrorIni** specifies the initial prediction error value of a new classifier.
- **fitnessIni** specified the initial fitness value of a new classifier.

5 The Structure of the Output

The XCSJava program produces always an output of the performance in the environment in that it is applied in. The output is written to the file specified in the second argument when starting the program.

Every 50 problems the XCS code produces some output using the 'writePerformance' function. Each line in the output represents the performance at a certain point in time. The different values are separated by a blank.

In a single-step environment the XCSJava program writes:

- The number of the so far examined problems.
- The percent of correctly solved problems in the last 50 problems.
- The error in the predictions averaged over the last 50 problems.
- The current size of the population in macro-classifiers.

In a multi-step environment the 'writePerformance' function writes out:

- The number of the so far examined problems.
- The number of steps to the food averaged over the last 50 exploitation trials.
- The error in the predictions averaged over all steps in the last 50 exploitation trials.
- The current size of the population in macro-classifiers.

6 A few Comments

The code is distributed for academic purposes without any warranty, either expressed or implied, to the extend permitted by applicable state law. We are not responsible for any damage resulting from its proper or improper use. By saying that, we want to emphasize that the code was extensively tested and we hope that no more mistakes are included.

If you have any comments or suggestions or identify any bugs, please contact the author.

7 Acknowledgments

The author would like to thank the Automated Learning Group of NCSA at UIUC for support and useful discussions. Furthermore, the author would like to thank Stewart Wilson for the great correspondence as well as Martin Pelikan and Kumara Sastry for valuable discussions and useful comments.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants F49620-94-1-0103, F49620-95-1-0338, F49620-97-1-0050, and F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. Support was also provided by a grant from the U. S. Army Research Laboratory under the Federated Laboratory Program, Cooperative Agreement DAAL01-96-2-0003. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are my own and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air

Force Office of Scientific Research, the National Science Foundation, the U. S. Army, or the U.S. Government.

References

Butz, M. V., & Wilson, S. W. (2000). *An algorithmic description of XCS* (IlliGAL report 2000017). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

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

Wilson, S. W. (1998). Generalization in the XCS classifier system. In Koza, J. R. e. a. (Ed.), *Genetic Programming 1998: Proceedings of the third annual conference* (pp. 665–674). San Francisco: Morgan Kaufmann.

Wilson, S. W. (1999). Get real! XCS with continuous-valued inputs. In Booker, L., Forrest, S., Mitchell, M., & Riolo, R. L. (Eds.), *Festschrift in Honor of John H. Holland* (pp. 111–121). Center for the Study of Complex Systems. http://prediction-dynamics.com/.