An Approach to Analyze the Evolution of Symbolic Conditions in Learning Classifier Systems

Pier Luca Lanzi, Stefano Rocca, Stefania Solari
Artificial Intelligence and Robotics Laboratory (AIRLab)
Politecnico di Milano. I-20133, Milano, Italy
pierluca.lanzi@polimi.it,rocca.ste@gmail.com,solari.stefania@gmail.com

ABSTRACT

In this paper, we introduce an approach for the identification of building blocks in symbolic expressions and apply it to analyze the emergence of building blocks in XCS with symbolic representation. The objective is to extract from a sequence of evolving populations a set of recurrent patterns which identifies pieces of the problem solution, so to track the emergence of the optimal solution. This permits the introduction of better measures of performance which might be useful in diagnosing problems and adapting algorithms.

Categories and Subject Descriptors

F.1.1 [**Models of Computation**]: Genetics Based Machine Learning, Learning Classifier Systems,Genetic Programming

General Terms

Algorithms, Representation.

Keywords

LCS, XCS, XCSGP, Representation, Symbolic Conditions.

1. INTRODUCTION

Generalization is an important feature of learning classifier systems which heavily relies on the representation of classifier conditions. Among the several representations introduced in the literature, symbolic conditions are probably the most general but also the most computational demanding solution. Symbolic conditions are also daunting to study. The bloat that affects this type of representation [3] and the higher number of classifiers required makes the analysis of this type of systems more difficult [5]. This is also worsened by the lack of a well established approach to analyze GP-based populations [5]. In this paper, we introduce an approach for the identification of building blocks in symbolic expressions and apply it to analyze the emergence of building blocks in XCS with symbolic representation [3]. The objective is to extract from a sequence of evolving populations a set of recurrent patterns which identifies pieces of the problem solution, so to track the emergence of the optimal solution. This permits the introduction of better measures of performance which might be useful in diagnosing problems and adapting algorithms. To achieve this we (i) trace the evolutionary process by keeping a sequence of evolving populations; (ii) for each population we simplify individuals into the "canonical form"; (iii) we count the occurrences of each subexpression appearing in such simplified individuals; finally, (iv) we analyze the emergence of subexpressions during the evolutionary process. If a subexpression has an increasing number of occurrences during the evolutionary process, it means that, fitness-wise, it has been considered useful to reach the optimal solution and it shall be considered a building block.

2. THE EXTRACTION PROCEDURE

Given a sequence of populations representing the evolution of a solution, our approach transforms each individual in the populations in a *canonical form* so that two syntactically different but semantically equivalent individuals are represented by the same *canonical form.*

Such simplified individuals are then analyzed to extract recurrent patterns, i.e. recurrent subexpressions; we consider highly recurrent patterns as building blocks. The detailed descriptions of the procedures discussed here are available in [5].

2.1 The Canonical Form

Individuals are transformed into the "canonical form" through four steps: (i) simplification, (ii) conversion into the disjunctive normal form, (iii) normalization of the expressions that compose the conjuncts, and (iv) sorting of the expressions that compose the conjuncts.

Simplification. In order to reduce the amount of useless and redundant code, we first simplify the individuals before extracting the subexpressions. To simplify an individual it is possible to define some simplification rules or specialized operators [2]. In our work, simplification is based on Mathematica [4] that exposes, through a library, a set of functions to simplify mathematical expressions which can be linked and then invoked directly in another program. This initial simplification step gets rid of all the useless and redundant code to bring the expression into a more compact form. For example, the expression "$X0 + X1 + (1-1)X0 - 10 > -5$ AND $5(X0 - X0) = 0$" will become "$X0 + X1 - 5 > 0$".

Conversion into disjunctive normal form. If Boolean operators are included, individuals are then expanded in order to obtain the disjunctive normal form. The disjunctive normal form of an expression is a disjunction (sequence of ORs) consisting of one or more disjuncts, each of which is a conjunction (AND) of one or more Boolean expressions.

Normalization. Elementary expressions obtained through the previous steps are Boolean expressions that compose the conjuncts of individuals in disjunctive normal form. Although all the expressions have been subjected to simplification, two elementary expressions can still be syntactically different but semantically equivalent. For example, the expressions "$2(X0 + X1) < 9$" and "$X1 + X0 < 9/2$" cannot be simplified anymore, but they are semantically equivalent, but syntactically different. To solve this problem we developed a procedure that rearranges an elementary expression into a normalized form such that two equivalent elementary expressions are syntactically equivalent (see [5] for details). This routine subtracts the second term of the equality or inequality to both sides of the equality/inequality forming a second term that is 0. Then it expands out products and positive integer powers in the first term, and divides both sides by the first coefficient of the first term.

Sorting. Two subexpressions composed by exactly the same set of elementary expressions conjuncted in different order are still semantically equivalent but syntactically different. For example, consider the expressions "$X0 > 0\ AND\ X1 > 3$" and "$X1 > 3\ AND\ X0 > 0$". Although these two expressions are semantically equivalent they are syntactically different. To solve this problem it is possible to extract the elementary expressions splitting each expression around "*AND*", alphabetically sort the obtained set, and then reconnect the elementary expressions to form a new subexpression. Applying this procedure to the two subexpressions just shown, we obtain the same expression "$X0 > 0\ AND\ X1 > 3$".

2.2 Population Simplification

Given a population generated by XCS with GP-based conditions, classifier conditions are reduced to the previously described "canonical form". For this purpose we use a data structure M_s, a set of pairs composed of the following elements: (i) an elementary expression, obtained by splitting a subexpression around "*AND*", and (ii) the corresponding semantically equivalent elementary expression that is used to represent all the elementary expressions semantically equivalent to the first element of the couple. The structure M_s maps generic expressions to their normalized form. This mapping is kept for two reasons: (i) it assures that in each set of subexpressions built using the same M_s semantically equivalent subexpressions have the same representation; (ii) if an elementary expression has already been found it is not necessary to recompute its normalized form.

2.3 Extraction of Subexpressions

We extract the subexpressions appearing in the population by splitting the classifier conditions, expressed in the canonical form, around certain operators. The issue in this case is how to decide around which operators we should split conditions. We need subexpressions that represent nuggets of interesting information about the problem solution. As an example, consider the problem of learning the inequality "$X0 + X1 > 4$" with $X0, X1 \in \mathbb{N}$. Suppose we have the individual "$3X0 > 10$". If we split the expression "$3X0 > 10$" around the "$>$" symbol we obtain two subexpressions that do not represent any interesting information about the solution. The same happens if we split a subexpression like "$X1 * (X1 + X2)$" around "$*$": in this case each of the two subexpressions defines a sort of "context" for the other. Furthermore consider the expression "$X1 > 3\ AND\ NOT(X0 = 0)$". If we split this expression around the "*AND*" clause we obtain the two subexpressions "$X1 > 3$" and "$NOT(X0 = 0)$". In this case the second subexpression does not represent interesting information about the problem solution. This happens because also in this case each subexpression defines a sort of "context" for the other. Finally consider another expression, that is "$X1 > 3\ AND\ NOT(X0 = 0)\ OR\ X0+X1 > 4$". If we split this expression around "*OR*" we obtain the two subexpressions "$X1 > 3\ AND\ NOT(X0 = 0)$" and "$X0 + X1 > 4$". Both of them represent very interesting information about the problem solution. This is because two subexpressions joint by "*OR*" are independent. So we choose to extract subexpressions splitting individuals around "*OR*". Furthermore individuals are represented by a disjunctive normal form, so there is no problem in doing it.

3. THE ANALYSIS PROCEDURE

After the population has been examined and all the subexpressions which appear in it have been extracted, we count the occurrence of each subexpression in the population. The output of this phase, for a population P_t, is a set of triplets S_t each one composed of the following elements: (i) a subexpression, (ii) the frequency of the subexpression in the population P_t, and (iii) the set of the individuals in P_t that contain subexpressions that are semantically equivalent to the first element of the triplet. Applying the algorithm to the populations we obtain a set of S_t. Two semantically equivalent subexpressions that belong to two different S_t are syntactically equivalent. Because of this property it is possible to analyze the trend of the frequency of a subexpression in all the P_t of the experiment. By applying this technique to a sequence of evolving populations it is possible to analyze the trend of the frequency of the subexpressions appearing in the population. Subexpressions that have an increasing trend are considered useful, while subexpressions that have a constant or decreasing trend are considered useless or too specific. Subexpressions with an increasing trend may viewed as building blocks.

4. EXPERIMENTS WITH XCSGP

We applied the proposed methodology to analyze the evolution in XCS with GP-based conditions [3]. Note that, since XCS uses macroclassifiers, the analysis of subexpression frequency takes into account numerosity [6]. In addition, since in XCS schemata are defined based both on the conditions and on the actions [1], the analysis of frequent subexpressions considers individuals that advocate the same action.

We considered the simple problem of learning the inequality "$X0 + X1 > 4$" with $X0, X1 \in \mathbb{N}$ and $X0, X1 \in [0, 9]$ (other problems are discussed in [5]). We applied XCSGP with a population of 1000 classifiers, all the other parameters are set as usual [6, 5]. Ten runs were performed, each one consisting of 50000 learning problems that XCSGP had to solve. During each run, the population was saved every 1000 problems, so at the end of each run we collected 50 populations. Here, we analyze two example runs: one consisting of successful convergence, the latter analyzes an example of run that did not reach the optimal solution.

In the first run XCSGP needed around 22000 learning problems steps to reach 100% performance (see [5]). However, 95% performance was reached by less than 10000 learning problems. We first analyze the classifiers that advocate action 0 which represent the negation of the inequality "$X0 + X1 > 4$", that is "$X0 + X1 < 5$". Figure 4 shows the trend of the frequencies of recurrent subexpressions. We have plotted the subexpressions with the highest average frequency. From the plot reported in Figure 4, it is possible to see that subexpressions that represent interesting information about the problem have an increasing trend, while useless subexpressions have a decreasing trend. To better analyze the trends of the frequency of the subexpressions we apply linear regression and obtain the plot reported in Figure 1. From this plot we extract the following recurrent subexpressions with an increasing trend: (i) "$X0^2 * X1 = 3$", (ii) "$9 * (X0 + X0 * X1 + X1^2) < 27 + 26 * X1$", (iii) "$5 * X0 < 14\ AND\ X1 < 3$", (iv) "$X0 * X1^2 = 3\ AND\ X1 < 3$", (v) "$X0 * X1^2 = 3$", and (vi) "$5 > X0\ AND\ X1 < 1$". The subexpression (iv) can be further simplified to "$X0 = 3\ AND\ X1 = 1$" because $X0, X1 \in \mathbb{N}$, but Mathematica was not able to do this simplification because we have not specified the variable domain. All these subexpressions represent interesting information about the problem and are all subsets of "$X0 + X1 < 5$", so all are correct. Some of these subexpressions, like (i) and (v), are very specific but are not included in the other subexpressions. This is the cause of their growth in spite of their specificity. The subexpression "$X0 < 2\ AND\ X1 < 3$" instead has a decreasing trend because it is a subset of (iii), that has an increasing trend. The subexpression "$1 + X1 > 2 * X0 * X1\ AND\ X1 < 9$ is not a subset of "$X0 + X1 < 5$" so it does not represent a correct mapping for the action 0. It is possible to note that it has a decreasing trend. Note that the union of the subexpressions with an increasing trend listed before is almost a complete mapping between variable values and the action 0: only "$X0 = 0\ AND\ X1 = 4$" is not included in any of these subexpressions but is included in (vii) "$9 * (X1^2 + X0 * (2 + X1)) < 54 + 26 * X1$". This last subexpression has an increasing trend but a low frequency because it appears only after step 43000. Furthermore (vii) includes (ii), but (vii) has not replaced (ii) because (vii) has appeared too late in the evolution.

We now analyze the classifiers that advocate action 1. They represent the inequality "$X0 + X1 > 4$". Figure 2 shows the trend of the frequencies of recurrent subexpressions. We have plotted the subexpressions with the highest average frequency. Also in this case it is possible to see that subexpressions that represent interesting information about the problem have an increasing trend, while useless subexpressions have a decreasing trend. From this plot we extract the following recurrent subexpressions with an increasing trend: (i) "$6 > X1\ AND\ X0 > 4$", (ii) "$7 > X0\ AND\ X1 * (-3 + X0 + X1) > X1$", and (iii) "$X1 * (-3 + X0 + X1) > X1$". Note that the subexpression (iii) is equivalent to "$NOT(X1 = 0)\ AND\ X0 + X1 > 4$". Also in this case, all the recurrent subexpressions with an increasing trend represent interesting information about the problem, none of them is wrong. The subexpression (ii) is contained in the subexpression (iii) but it is not replaced by it. Although it is possible to note that the quick growth of the subexpression (ii) has stopped around step 15000, with the emergence of the subexpression (iii), then (ii) has decreased and after step 30000 has an almost constant trend. So the more general subexpression (iii) has grown hindering (ii). It is possible to note that the subexpression (iii) is contained in many other subexpressions like "$X0 > 0\ AND\ X1 * (-3 + X0 + X1) > X1\ AND\ X1^3 > 2 * X1^2$" that has a slightly increasing trend. Subexpressions that are overly specific, like "$6 > X1\ AND\ X0 > 8$", have a decreasing trend. Note that in this case the union of the subexpressions with an increasing trend listed before is a complete mapping between variable values and the action 1.

The second run we report reached only a 90% performance by 25000 learning problem (the same problem can be reliably solved using a population of 1600 classifiers [5]). We then analyze the classifiers that advocate action 1. They represent the inequality "$X0 + X1 > 4$". Figure 5 reports the trend of the frequencies of recurrent subexpressions. Although the system has not reached optimal solution, it is possible to see that subexpressions that represent interesting information about the problem have an increasing trend, while useless subexpressions have a decreasing trend. This is because the average number of classifiers for each population that advocate action 1 is high, about 991. From the plot we note the following recurrent subexpressions with an increasing trend: (i) "$X0 > 0\ AND\ X1 > 7/2$", (ii) "$X0 > 4$", (iii) "$X1 > 6$", and (iv) "$X1 > 5$". Although the system did not reach optimal performance, all the recurrent subexpressions with an increasing trend listed before represent interesting information about the problem and they are all correct. Wrong subexpressions or subexpressions that are subset of others have a decreasing trend. For example, the subexpression "$X1 > 7/2$" has a decreasing trend and it is wrong because it is not a subset of "$X0 + X1 > 4$". Another example, consider the subexpression "$X0 > 7$": it has a decreasing trend and it is a subset of the subexpression (ii). It is possible to note that the quick growth of the subexpression "$X0 > 7$" has stopped around step 3000, with the emergence of the subexpression (ii), then the frequency of "$X0 > 7$" has decreased. The union of the increasing subexpressions listed before represents an almost complete mapping between variable values and the action 1, only "$X0 = 0\ AND\ X1 = 5$" is not contained in any of them.

5. SUMMARY

We have proposed a method to analyze the evolution of XCS with symbolic conditions. We have applied our approach to analyze two experiments involving the learning of a simple numeric expression. By applying the proposed method, we have been able to show how XCS with GP-based conditions, XCSGP, builds up an optimal solution by favoring the evolution of *good* subexpressions that are useful with respect to the target problem. Our approach can also be used in a constructive way: after the frequent subexpressions have been identified, they can also be used to guide the generation of offspring classifiers, as shown in [5].

6. REFERENCES

[1] Martin V. Butz, Tim Kovacs, Pier Luca Lanzi, and Stewart W. Wilson. Toward a theory of generalization and learning in xcs. *IEEE Transaction on Evolutionary Computation*, 8(1):28–46, February 2004.

[2] John R. Koza. Hierarchical automatic function definition in genetic programming. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 297–318, Vail, Colorado, USA, 24–29 1992. Morgan Kaufmann.

[3] Pier Luca Lanzi and Alessandro Perrucci. Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In Wolfgang Banzhaf et al., editor, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, pages 345–352, Orlando (FL), July 1999. Morgan Kaufmann.

[4] Wolfram Research. Mathematica 5. http://www.wolfram.com.

[5] Stefano Rocca and Stefania Solari. Building blocks analysis and exploitation in genetic programming. Master's thesis, April 2006. Master thesis supervisor: Prof. Pier Luca Lanzi.

[6] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.

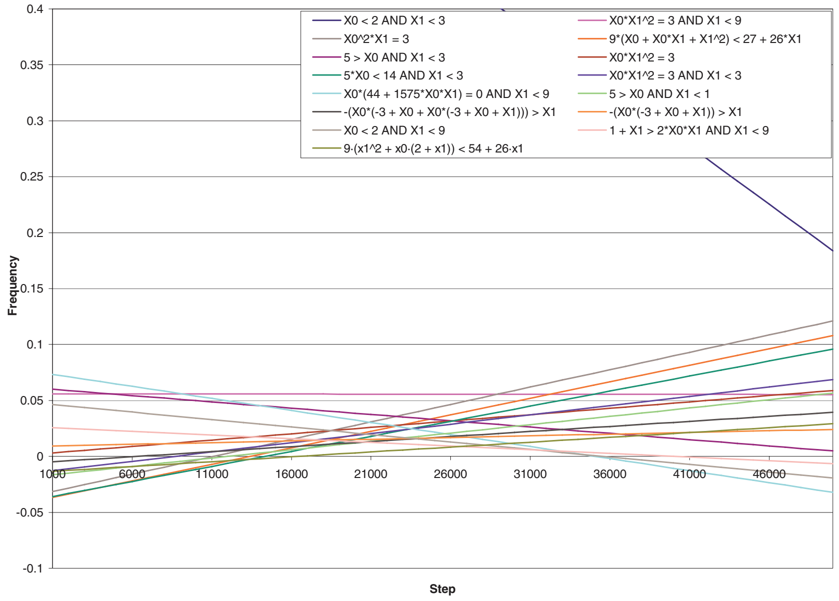

Figure 1: Linear regression of the frequency trends of the subexpressions for the classifiers with action 0.

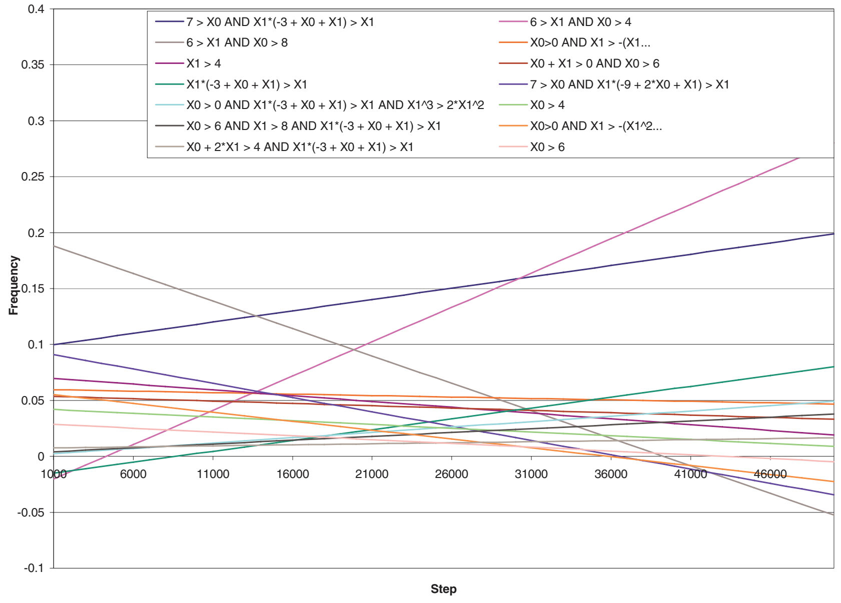

Figure 2: Linear regression of the frequency trends of the subexpressions for the classifiers with action 1.

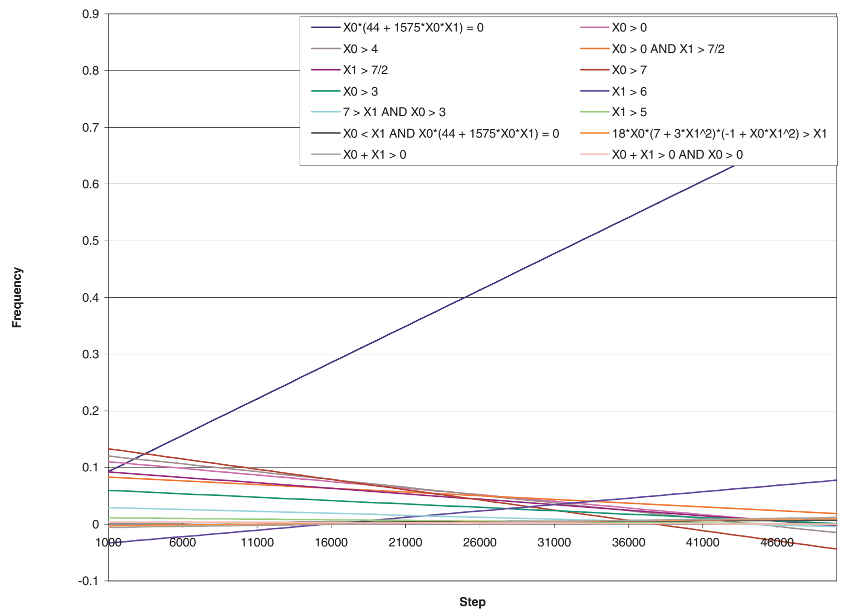

Figure 3: Linear regression of the frequencies of recurrent subexpressions in the second run when the classifier action is 0.

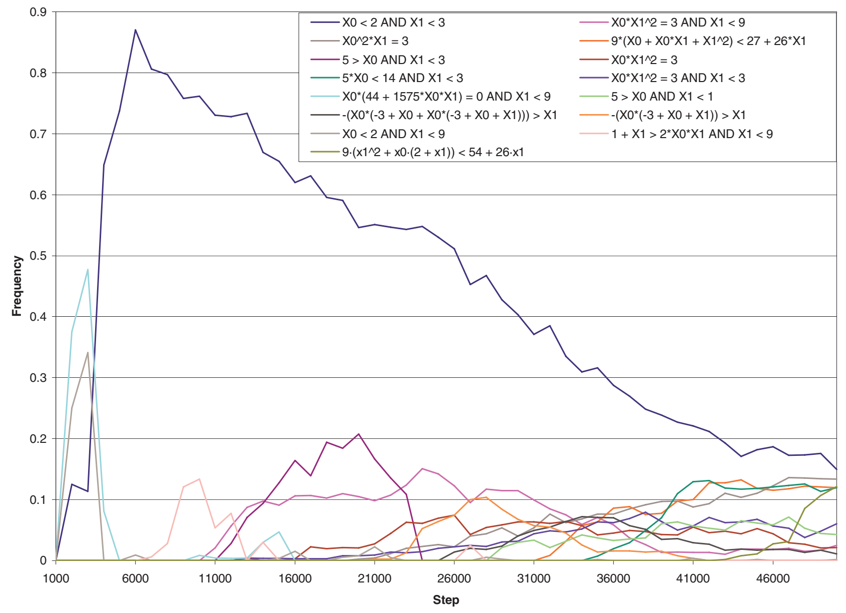

Figure 4: Frequency trends of the subexpressions for the classifiers with action 0.

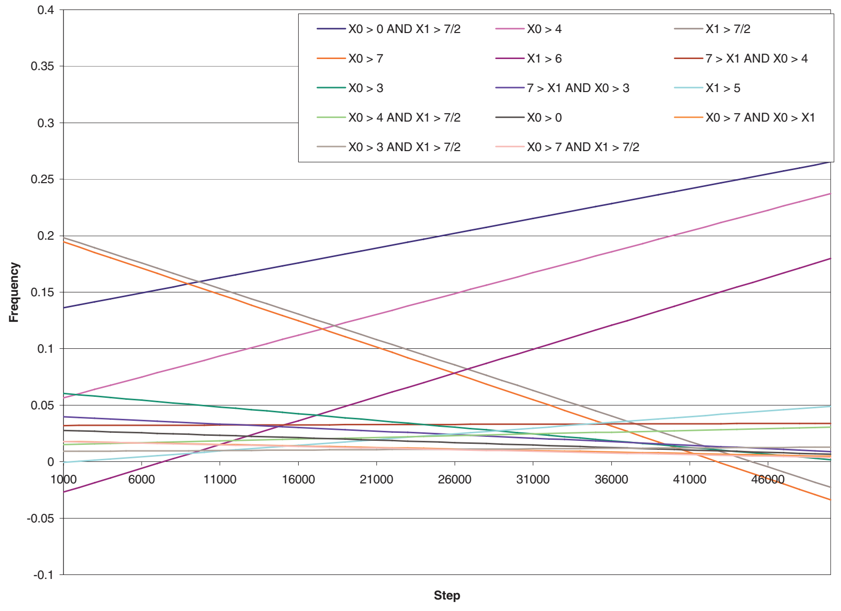

Figure 5: Linear regression of the frequencies of recurrent subexpressions for the second run when classifier action is 1.