Experimental Comparison of Symbolic Learning Programs for the Classification of Gene Network Topology Models

Andreas D. Lattner[1], **Sohyoung Kim**[2], **Guido Cervone**[2], **John J. Grefenstette**[2]

[1]Center for Computing Technologies – TZI, University of Bremen,
PO Box 330 440, D-28334 Bremen, Germany, adl@tzi.de

[2]George Mason University, 4400 University Drive, Fairfax,
Virginia 22030-4444, USA, {skime|gcervone|jgrefens}@gmu.edu

Abstract

This study addresses the problem of identifying the large-scale topology of gene regulation networks from features that can be derived from microarray data sets. Understanding large-scale structures of gene regulation is fundamentally important in biology. Recent analysis of network properties of known biological networks has shown that they display scale-free features, but it is not yet clear whether the scale-free features are generic to all biological networks. In this work five different symbolic classifiers – AQ20, C4.5, C4.5rules CN2 and RIPPER – were employed to classify simulated networks as random or scale-free. In the best case, an average accuracy of over 96% could be achieved by C4.5rules in ten cross-validation runs.

1 Introduction

Understanding the large-scale structures of gene regulation networks provides insights about universal biological structural design principles and a better understanding of dynamical processes of gene regulation. Gene regulatory networks are connected structures between regulator genes and target genes. The genes in these networks interact to regulate their expression levels.

Recently, there has been growing interest in network structure models that have specific design principles and their inherent statistical properties [Albert and Barabási, 2002]. Three main classes of network models, exponential networks [Erdős and Rényi, 1959], scale-free networks [Barabási and Albert, 1999], and small-world networks [Watts and Strogatz, 1998], have been used to describe topological features of various naturally occurring systems including the Internet, social networks, and some biological networks [Albert and Barabási, 2002]. Recent analyses of network properties of the metabolic maps [Jeong *et al.*, 2000], molecular interaction networks [Fox and Hill, 2001], and protein-protein interaction networks [Jeong *et al.*, 2001] have shown that these biological networks display scale-free features.

Scale-free networks are characterized by a connectivity distribution which follows power-law distribution. The connectivity between nodes is very heterogeneous, i.e., there are many nodes with few connectivity and a small number of nodes with high connectivity. Compared to scale-free networks, in the random network model the connectivity distribution follows a Poisson distribution, i.e., the number of edges between nodes is quite even and nodes with high connectivity are not very likely to occur.

The observations of scale-free features are encouraging because the model provides important features that might be especially beneficial to biological systems [Jeong *et al.*, 2001; Albert *et al.*, 2000]. Knowing an appropriate model for biological networks will allow us to understand the following statistical features of the model. Particularly, high tolerance of scale-free networks for the random node failures in terms of structural integrity might suggest that network topology partially accounts for the robustness of biological systems.

Although recent analyses of network properties of known biological pathways or networks have shown that the most examples of biological networks display scale-free features [Jeong *et al.*, 2000; Fox and Hill, 2001; Jeong *et al.*, 2001; Bhan *et al.*, 2002; Ravasz *et al.*, 2002], it is not clear yet whether the scale-free features are generic to all biological networks, due to the limited available information about pathways and connectivity. To overcome these limitations, there have been attempts to understand topology by inferring the underlying connectivity of gene networks from microarray time series data [Bhan *et al.*, 2002].

However, the inference of network connections from microarray data is a very difficult problem with current microarray data without further exhaustive experimental verification steps [Kitano, 2002]. Initial work applied a neural network to classify network structures from microarray data based on measured global characteristics without inferring individual connectivity between biological molecules [Kim *et al.*, 2003]. In this work the neural network successfully classified the data with a predictive accuracy of ca. 90%.

This work follows up [Kim *et al.*, 2003] by applying a series of symbolic machine learning programs to the same data. The motivation for this approach is that unlike neural networks, symbolic methods provide answers in a form that can be understood and validated by domain experts. The main goal of this work is therefore trying to understand if there are distinctive patterns that are characteristics of the two classes, and whether such patterns have a biological meaning, or are simply artifacts of the data.

2 Input Data

The data consisted of 2400 events divided into two classes (scale-free and random) of 1200 events each. Each event contained 20 continuous attributes plus a binary class attribute, and no missing values. The attributes represent 20 bins of gene frequencies and the binary class stands for scale-free or random network.

The data was generated using a gene regulation network

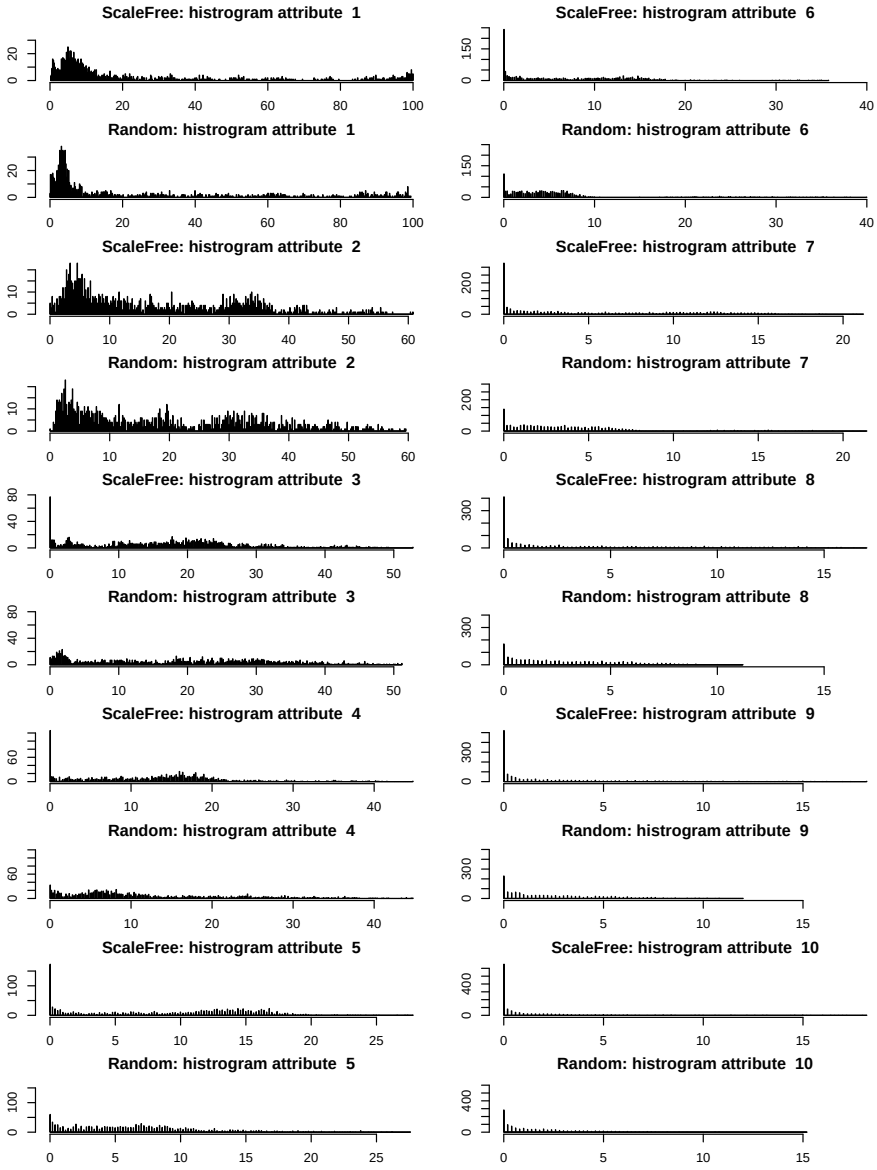

Figure 1: Histogram of attribute values, attributes 1 - 10

simulator. It allows the simulation of Boolean networks with arbitrary number of genes [Kim *et al.*, 2003]. In these experiments two network types (scale-free and random) and five different connectivity rates (1 - 5) were taken into account. For each of these ten topology/connectivity combinations 20 different networks with 500 genes were generated randomly. For all 200 networks perturbation propagation distributions were computed using 12 different perturbation rates (1%, 3%, 5%, ..., 19%, 21%, 40%) resulting in 2400 histograms. The procedure to create the perturbation propagation histograms is the following:

1. Randomly initialize the gene network, represented by an array of genes t_1. Determine the status t_2 of the gene network in the next time step by applying the transition function.
2. The following steps are repeated 100 times:
   - Apply perturbation to a certain percentage of genes (specified by the perturbation rate) and store the new gene network in t_1'.
   - Apply the transition function to t_1' and store the gene network of the next time step in t_2'.
   - Compare the genes of t_2 and t_2' and accumulate the number of discrepancy for each gene.
3. After 100 runs for each gene the frequency of perturbation is known. Now different bins collect the frequencies of genes with certain perturbation frequencies. The 20 different bins sums up all genes that show up 0 - 5%, 5 - 10%, ..., 95 - 100% perturbation through the time series, where the first bin represents genes with lowest perturbation frequencies and the last (twentieth) bin represents genes with highest perturbation frequencies.

Each histogram with the 20 different bins represents one training (or testing) event. The attributes represent different bins, i.e., the frequency of genes with certain perturbation frequencies. A more detailed description of the data generation can be found in [Kim *et al.*, 2003].

Figures 1 and 2 summarize the input data in histograms. These histograms do not show the single event histograms, which were used for training and testing, but the value distributions for all scale-free and random model events for each attribute. Each attribute corresponds to one of the

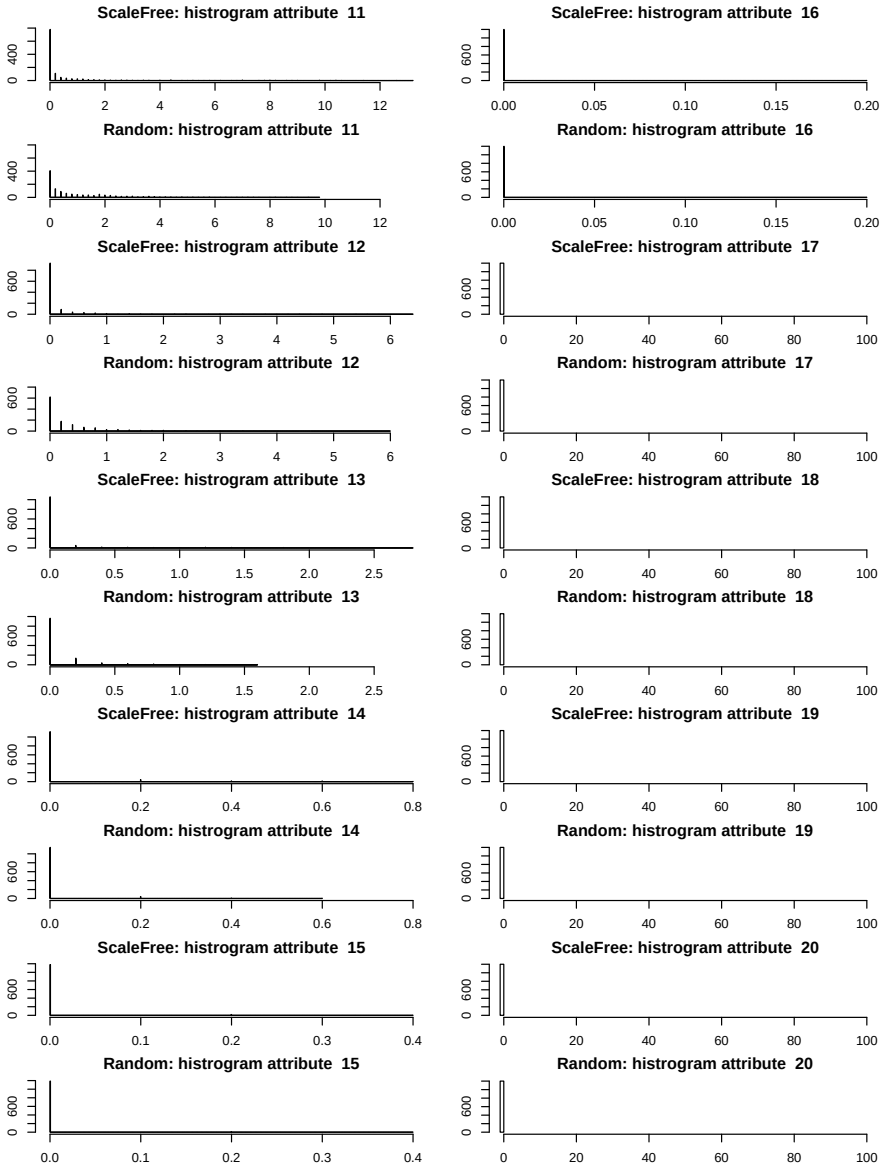

Figure 2: Histogram of attribute values, attributes 11 - 20

twenty bins mentioned before. The x-axis represents all values of the attribute and the y-axis shows the number of events with the corresponding value.

In the graphs no trivial separation between the two classes can be found for single attributes. Nevertheless, for some attributes differences in the frequency of values can be seen. E.g., for attribute 1 the random model has a higher peak for lower values than the scale-free model. The scale-free model leads to more values near zero than the random model for attribute 3, 4, and 6. At attribute 5 events of the scale-free model tend more to concentrate between the values 10 and 18 than the ones from the random model.

Another interesting observation is that the two models seem not to differentiate regarding attributes 15 - 20. Especially for attributes 17 - 20 all events have the value zero. The explanation for that is, that these bins represent frequencies of highly perturbed genes which do not occur in this test data. Thus, attributes 17 - 20 could be left out in experiments without impairing the results.

Figure 3 shows the distributions of the attributes 1 - 14 and their pairwise correlations. The figure is symmetric, i.e., for each image above the diagonal exists a counterpart underneath it. The images with the pairwise correlations plot all instances in two-dimensional graphs where the x-axis represents the attribute in the current column and the y-axis represents the attribute in the current row. The diagonal shows the distributions for single attributes, i.e., how often certain attribute values appear in the data.

As the more important information seems to be in the first part of the attributes, we do not display the distributions of and correlations with attributes 15 - 20. As it can be seen the value distributions of attribute 2 - 5 are spread out more evenly than the others. The remaining attributes in this figure have a stronger tendency to lower values. The correlation images between the attributes show, that for some attribute combination parts of the events can be separated from each other, as it is the case with different combinations of attributes 8 - 13.

3 Experiments

In this section the different experiments on the data are described. After introducing the evaluation criteria and learning programs the results of the machine learning programs

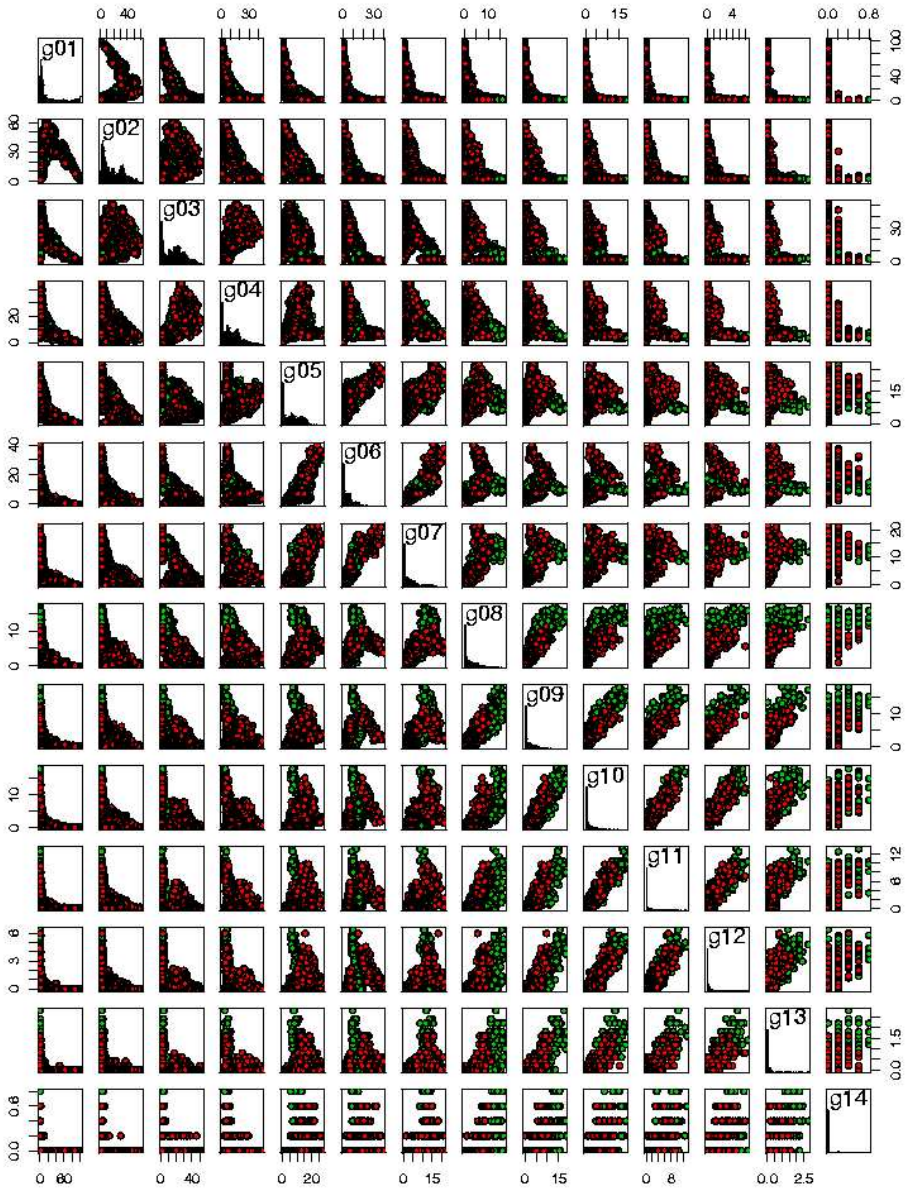

Figure 3: Distributions and pairwise attribute correlations for first 14 attributes

are presented.

3.1 Evaluation Criteria

The evaluation of the learned knowledge was done using different criteria. The most important is the predictive accuracy, which represents the number of correct classifications over the total number of classifications. In the case of AQ, this value must be paired with precision, since the program may assign more than one class to each event (see Section 3.2).

The learned knowledge is also compared in terms of time and complexity which is computed as the size of the tree for C4.5 and the number of rules and conditions for the other decision rule learning programs. Only AQ20 provides training and testing times. Among the other tools, RIPPER is the only one that writes its learning time to the output. To be able to compare durations of the programs, we used the GNU `time` command to get values for the system and user time used during learning and testing.

Different criteria were chosen to emphasize the strength and weakness of the algorithms. This is to reflect the fact some algorithms may perform better in terms of predictive accuracy, at the expense of a bigger learning time or rules of higher complexity.

3.2 Learning Programs

In this section the learning programs employed in this study are presented briefly. More detailed descriptions can be found in [Cervone *et al.*, 2001; Quinlan, 1993; 1996; Clark and Niblett, 1989; Cohen, 1995].

AQ20

AQ20 is an implementation of the AQ learning methodology, which traces its origin to the A^q algorithm for solving general covering problems of high complexity [Michalski, 1969a; 1969b].

An implementation of the AQ algorithm in combination with the variable-valued logic representation produced the first AQ learning program, AQVAL/1, which pioneered research on general-purpose inductive learning systems [Michalski, 1975].

AQ20 is a machine-learning environment that supports users in conducting machine-learning experiments. AQ is a separate and conquer algorithm, also called progressive covering. It learns hypotheses given a set of examples and counter-examples that are either 1) complete and consistent with the input data (theory formation mode), 2) that allow either certain negative events to be included in the learned hypotheses (complete and inconsistent), 3) that do not include all the positive events (incomplete and consistent) or 4) a combination of the last two (incomplete and inconsistent). The ability of learning rules that are incomplete and inconsistent is very important in data mining, because it makes the algorithm cope well with noise.

Some of the most important features of AQ20 for this work are: ability to work with continuous data without discretization, ability to learn hypotheses according to multi-criterion optimization functions, and ability to test learn hypotheses according to different events / ruleset matching scheme.

AQ20 provides a collection of different algorithms for rule learning, pre- and postprocessing. The program is still under development and not all of the possible settings were tested in our experiments. We applied the theory formation (tf) mode, which learns complete and consistent rules for the training events. The minimum number of unique covered events by a rule was set to 20 (minimum_u = 20), the maximum number of rules kept by a star was 20 (maxrule = 20), and the weight for specifying the tradeoff between completeness and consistency was set to 0.0 (w = 0.0). The beam search was limited to the star size 1 (max_star = 1).

The rules were tested with the ATEST methodology (see [Cervone *et al.*, 2001]). For the interpretation of the disjunctions of rules the rule with the maximum match was applied (aggregation = max) and for the interpretation of conjunction (within a rule) the selectors ratio was used, i.e., the match is determined by ratio of the number of satisfied conditions to all conditions in a rule (mode = selectors_ratio).

All other programs were used with their default settings.

C4.5 and C4.5rules

The decision tree learner C4.5 was developed by Quinlan [Quinlan, 1993]. It is based on the ID3 (Induction of decision trees) algorithm. In our experiments we used C4.5 Release 8, which is described in [Quinlan, 1996].

Given a set of training examples it generates a decision tree, that can be used to classify unknown instances. In the beginning all events of all classes are assigned to the root node of a tree. Now, recursively the tree is grown by selecting the attribute that leads to the best information gain to divide the set of events in a node into subsets. A node is not refined any further if it only consists of examples of one class or another stop criterion is satisfied.

C4.5rules is the decision rule counterpart of C4.5. It generates a set of rules for each path from a learned decision tree. Then it is checked if the rules can be generalized by dropping conditions. In another step the most useful subset of the rules is selected.

CN2

The CN2 algorithm was developed by Clark and Nibblet to learn a set of propositional rules given a set of examples and counter examples [Clark and Niblett, 1989].

This algorithm can be viewed as a hybrid of AQ learning and decision tree learning. It uses beam search as the AQ algorithm, but it specializes rules in the same fashion decision trees are built.

It first generates general rules which are then specialized by adding conditions until the learned rules have statistical significance over the training events. The process is repeated until all the training events have been covered.

CN2 can generate ordered or unordered decision rules.

RIPPER

Cohen developed the decision rule learner RIPPER by modifying the "incremental reduced error pruning" (IREP) method [Cohen, 1995]. The IREP method by Fürnkranz and Widmer [Fürnkranz and Widmer, 1994] integrated pre-pruning and post-pruning into learning.

IREP uses a separate-and-conquer algorithm to create rulesets. It creates one rule at time and removes all positive (and negative) examples covered by this rule from the example set. To create a rule the uncovered examples are randomly split into a growing set and a pruning set. At rule generation it is started with an empty conjunction of conditions. The rule is grown by rapidly adding the condition that maximizes the FOIL information gain criterion. After growing the rule it is immediately pruned [Cohen, 1995].

In the "repeated incremental pruning to produce error reduction" (RIPPER) for each rule in the ruleset two rules are generated: the "replacement" and the "revision" which are formed by growing and pruning to minimize the error of the complete ruleset. Then it is decided to keep the old rule or replace it by one of its two modifications by using the minimum description length (MDL) heuristic. This optimization step can also be repeated by RIPPER.

The pruning set is used during the pruning of the rules. The deletion of conditions of a rule depends on how many positive and negative examples from the pruning set are covered after the modification of a rule.

3.3 Results

A number of experiments was performed on experimental data generated using the method presented in [Kim *et al.*, 2003]. In all experiments a cross-validation with ten splits was performed, i.e., in each of the ten runs one split was used for testing (10%) and nine splits were used for training (90%). All experiments were run on a Pentium 4 (mobile), 1.8 GHz, 512 MB RAM, under Red Hat Linux 9.

The results of the different programs are compared in Table 1. All values in the table are averages of ten cross-validation runs and the standard deviation ($average\ value \pm stddev$). To better compare the algorithms, we used identical splits to train and test all the programs.

The generated ruleset were tested using each method's own testing procedure. In the case of AQ20, it was more difficult to compare the results because this program may assign more than one class to each event. The AQ20 testing procedure generates degrees of match between events and rulesets. An event is classified as belonging to the class whose ruleset achieve the highest score. If more than one class obtains the same score, or a score within a certain tolerance, then the event is classified as belonging to all the classes with top score. The event classification in these

Table 1: Results of the different programs in ten cross-validation runs

	AQ20	C4.5	C4.5rules	CN2	Ripper
Predictive accuracy (single) in %	(92.2)	95.69±1.56	96.33±1.49	90.78±2.28	95.669±1.54
Predictive accuracy (multiple) in %	94.793±1.97	N/A	N/A	N/A	N/A
Precision (multiple) in %	94.995±2.59	N/A	N/A	N/A	N/A
Total time in s	9.06±0.690	0.18±0.01	0.246±0.02	2.697±0.27	1.474±0.19
# rules	17.77±1.92	N/A	27.8±1.99	38.1±3.67	13.3±2.26
# conditions / size of tree	159.11±19	101.6±6.11	87.5±5.93	95.6±8.58	54.3±7.68

cases is counted as correct if the correct class is among the assigned classes.

Because of the generation of multiple answers for each event, a new measure called precision was introduced. Precision is defined as: $\frac{(e*c-d)}{d(c-1)}$ where e is the number of events that matched at least one of candidate rulesets with a degree of match equal to or above the acceptability threshold, c is the number of decision classes, and d is the number of specific decisions (unique or alternative). The precision is 1 when for each testing event only one specific decision is returned, and 0 when for each testing event all possible decisions are returned as alternatives.

When the precision is 100%, the result of the predictive accuracy can be directly compared to the one of other methods.

In the experiments with AQ two classes were assigned to 62 testing events (out of 2400 events in all runs), thus its predictive accuracy can be almost directly compared to the regular predictive accuracies of the other tools. If all multiple classification results had been counted as false classifications (worst case), the average accuracy of the ten runs would have been $\frac{2213}{2400} = 92.2\%$ instead of $\frac{2275}{2400} = 94.79\%$.

The best accuracies have been achieved by C4.5rules. It correctly classified 96.33% of the events on average. C4.5 and Ripper almost have the same accuracies (95.69% and 95.67%), followed by AQ20 (92.2%) and CN2 (90.78%).

C4.5 and C4.5rules are the fastest (0.18 and 0.246 s), followed by Ripper and CN2 (1.474 and 2.697 s). AQ20 took most time (9.06 s).

Regarding complexity, Ripper generated the most compact rulesets (13.3 rules with 54.3 conditions on average). Except AQ20 (17.7 rules with 159.11 conditions), the other tools created more than the double number of rules (C4.5rules: 27.8 rules and 87.5 conditions, CN2: 38.1 rules and 95.6 conditions). The decisions trees learned by C4.5 had 101.6 nodes on average.

Figure 4 shows the usage of the attributes in the rules created by the different programs. As expected, attributes 17 - 20 have never been used, because they do not carry any information (in this data).

Sample outputs of the programs can be found in Appendix A. In all cases rules from the first cross-validation run are shown.

Specifically, C4.5rules showed the best accuracy with 96.33%. The decision trees of C4.5 and the rulesets generated by RIPPER performed almost the same. In these experiments RIPPER's and AQ20's rulesets have a much smaller complexity than the ones created by the other programs. With 13.3 rules RIPPER created less than half of the number of rules generated by C4.5rules and CN2. The rules of RIPPER also consist of much less conditions.

All the learning and testing times of the programs were very fast (< 3 seconds each run), except for AQ20, which took ca. 9 seconds for training and testing. C4.5 outperformed the other programs with the shortest duration of 0.18s per run on average.

The generated rulesets emphasize a big advantage of symbolic learning: the learned classifiers are comprehensible and can thus be evaluated by domain experts.

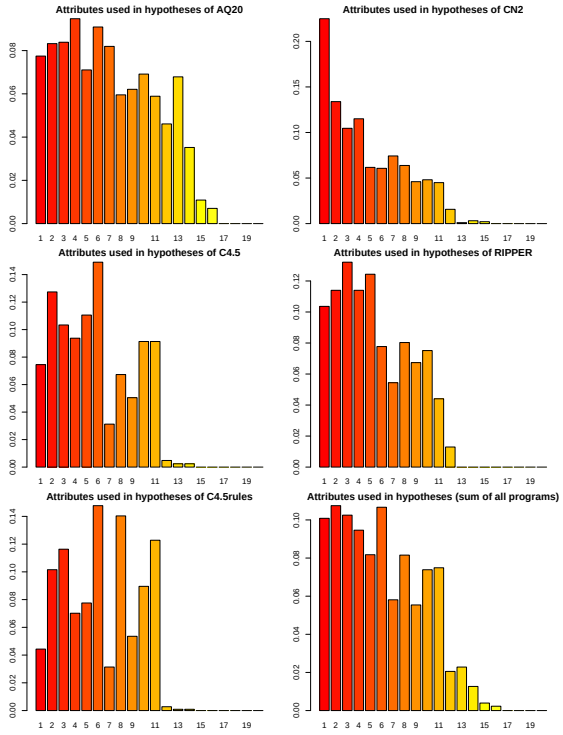

Figure 4: Attributes used by the different programs

4 Conclusion

Unlike previous methods for the inference of gene regulation networks from ideal data, this approach attempts to deal with data that is likely to be available from current experimental methods, such as microarray experiments.

As the results of the learning programs show, these features are suitable to distinguish between two different classes of network models, random and scale-free networks. The results of the experiments are very promising as some of the learning programs classified over 95% of the testing events correctly on average.

References

[Albert and Barabási, 2002] R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[Albert *et al.*, 2000] R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000.

[Barabási and Albert, 1999] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[Bhan *et al.*, 2002] A. Bhan, D. J. Galas, and T. G. Dewey. A duplication growth model of gene expression networks. *Bioinformatics*, 18:1486–1493, 2002.

[Cervone *et al.*, 2001] G. Cervone, L. A. Panait, and R. S. Michalski. The development of the AQ20 learning system and initial experiments. In *Proceedings of the 10th International Symposium on Intelligent Information Systems*, Zakopone, Poland, June 2001.

[Clark and Niblett, 1989] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[Cohen, 1995] W. W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, Lake Taho, California, 1995.

[Erdős and Rényi, 1959] P. Erdős and A. Rényi. *On random graphs I.* Publ. Math. (Debrecen), 1959.

[Fox and Hill, 2001] J. J. Fox and C. C. Hill. From topology to dynamics in biochemical networks. *Chaos*, 11:809–815, 2001.

[Fürnkranz and Widmer, 1994] J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In *Machine Learning: Proceedings of the Eleventh Annual Conference*. Morgan Kaufmann, New Brunswick, New Jersey, 1994.

[Jeong *et al.*, 2000] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabási. The large-scale organization of metabolic networks. *Nature*, 407:651–654, 2000.

[Jeong *et al.*, 2001] H. Jeong, S. P. Mason, A.-L. Barabási, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411:41–42, 2001.

[Kim *et al.*, 2003] S. Kim, J. N. Weinstein, and J. J. Grefenstette. Inference of large-scale topology of gene regulation networks by neural nets. In *Proceedings of the 2003 IEEE International Conference on Systems, Man & Cybernetics*, Washington, D.C., USA, October 2003. To appear.

[Kitano, 2002] H. Kitano. System biology: A brief overview. *Science*, 295:1662–1664, 2002.

[Michalski, 1969a] R. S. Michalski. On the quasi-minimal solution of the general covering problem. In *Proceedings of the V International Symposium on Information Processing (FCIP 69)*, volume A3, pages 125–128, Bled, Yugoslavia, October 8-11 1969.

[Michalski, 1969b] R. S. Michalski. Recognition of total or partial symmetry in a completely or incompletely specified switching function. In *Proceedings of the IV Congress of the International Federation on Automatic Control (IFAC)*, volume 27, pages 109–129, June 16-21 1969.

[Michalski, 1975] R. S. Michalski. Synthesis of optimal and quasi-optimal variable-valued logic formulas. In *Proceedings of the 1975 International Symposium on Multiple-Valued Logic*, pages 76–87, Bloomington, Indiana, May 13-16 1975.

[Quinlan, 1993] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[Quinlan, 1996] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[Ravasz *et al.*, 2002] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási. Hierarchical organization of modularity in metabolic networks. *Science*, 297:1551–1555, 2002.

[Watts and Strogatz, 1998] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

A Sample outputs of the learning programs

The following rulesets and the decision tree are the outputs of the different learning programs for the first of the ten cross-validation runs. The original outputs have been slightly changed for a more compact presentation. The attributes g1 - g20 represent the twenty different bins in the histograms of [Kim *et al.*, 2003].

Sample rules of AQ20

```
[class=0]
   <-- [g4<=29.5] [g5=1.7..7.7] [g9=0.1001..7.3]
       [g10>=0.1001] [g13<=0.9]
        : p=380,np=380,n=0,npq=1,u=24,cx=29,c=1 # 56747
   <-- [g3<=32.7] [g4<=19.3] [g5<=9.3]
       [g6=0.5..10.1] [g7=0.3001..8.7] [g8=0.7001..8.2]
       [g9>=0.1001] [g10<=11.9] [g13<=1.2] [g14<=0.4]
        : p=330,np=4,n=0,q=1,npq=1,u=20,cx=56,c=1 # 56763
   <-- [g1<=5.5] [g4<=34.9] [g5<=18.8]
       [g6<=8.1] [g7<=11.9] [g8=0.3001..8.3] [g9>=0.3001]
       [g10>=0.1001] [g11<=7.8] [g13<=1.3] [g14<=0.4]
        : p=315,np=214,n=0,npq=1,u=58,cx=57,c=1 # 56748
   <-- [g2<=54.1] [g3=1.7..36.9] [g4<=11.7]
       [g5<=22.7] [g6<=34.7] [g7<=20.7]
       [g8=0.3001..5.5] [g9<=3.5] [g13<=0.5999]
       [g14<=0.4] [g15<=0.09995]
        : p=312,np=41,n=0,q=1,npq=1,u=25,cx=59,c=1 # 56753
   <-- [g1<=4.1] [g2<=11.5] [g3<=45.9] [g4<=40]
       [g5=3.1..19.4] [g6=1.3..32.8] [g7<=15.7] [g8<=7.5]
       [g9<=8.3] [g11>=0.1001] [g13<=1.1]
        : p=187,np=50,n=0,q=1,npq=1,u=49,cx=59,c=1 # 56752
   <-- [g1>=0.8] [g2<=55.7] [g3<=15.7]
       [g4<=5.5] [g5<=18.9] [g6<=24.1]
       [g7=0.1001..16.2] [g8<=8.6] [g9<=13.1]
       [g13<=0.7] [g14<=0.4] [g15<=0.09995]
        : p=132,np=104,n=0,npq=1,u=55,cx=62,c=1 # 56749
   <-- [g1=3.9..97.2] [g2<=15.1] [g3=0.9001..25.3]
       [g4=0.1001..35.3] [g5<=14.3] [g6<=9.1]
       [g7<=7.3] [g8<=9.4] [g9<=8.4]
       [g10<=11.2] [g13<=1]
        : p=85,np=73,n=0,q=1,npq=1,u=67,cx=61,c=1 # 56751
   <-- [g1<=2.4] [g2<=3.5] [g5<=26.4]
       [g7<=20.4] [g8<=10.3] [g9>=2.6]
       [g11=1.3..10.1] [g13<=1.7] [g14<=0.5] [g15<=0.3]
        : p=84,np=79,n=0,q=1,npq=1,u=75,cx=52,c=1 # 56750
   <-- [g1>=48.8] [g2<=4.9] [g3=0.3001..21.4]
       [g4<=7.8] [g5<=4.3] [g6<=12.3] [g7<=6.8]
       [g8<=4.2] [g11<=4.9] [g13<=0.9] [g14<=0.2999]
        : p=27,np=23,n=0,q=1,npq=1,u=23,cx=57,c=1 # 56754

[class=1]
   <-- [g1=6.5..35.6] [g2<=30.7] [g3<=31.9]
       [g4<=35.6] [g5<=22.5] [g6<=25.1]
       [g7<=15.6] [g9<=4.9] [g10<=2.1]
       [g11<=0.7] [g12<=0.3] [g13<=0.09995]
        : p=277,np=277,n=0,npq=1,u=49,cx=62,c=1 # 113357
   <-- [g1=2.9..31.9] [g2=2.8..35.3] [g5=4.7..15.3]
       [g6>=2.9] [g9<=2.7] [g10<=0.5]
       [g11<=1.2] [g12<=0.3] [g13<=0.09995]
        : p=243,np=25,n=0,q=1,npq=1,u=25,cx=51,c=1 # 113367
   <-- [g1=2..48.9] [g2=7.1..51.2] [g3<=44.8]
       [g4=6.3..36.3] [g5>=2.9] [g7<=13]
       [g9<=0.5] [g10<=0.09995] [g11<=0.09995] [g12<=0.09995]
        : p=225,np=123,n=0,npq=1,u=78,cx=56,c=1 # 113360
   <-- [g1=1.1..35.6] [g2=2.7..30.7] [g3=2..20.1]
       [g4<=18.7] [g6<=20.7] [g7<=17.8]
       [g11<=9.2] [g14<=0.5]
        : p=223,np=179,n=0,npq=1,u=71,cx=46,c=1 # 113358
   <-- [g1=8.1..60.3] [g2>=15.8] [g3=4.1..37.5]
       [g4<=26.8] [g6<=1.1] [g7<=0.5]
       [g8<=0.09995] [g9<=0.09995] [g10<=0.09995]
       [g11<=0.09995] [g12<=0.09995] [g13<=0.2]
        : p=152,np=149,n=0,npq=1,u=123,cx=64,c=1 # 113359
   <-- [g1=3.1..45.8] [g2<=13.3] [g3<=46.7]
       [g4<=40.7] [g5<=20.6] [g6<=20.9]
       [g7<=14.2] [g9<=2.7] [g10<=1.7]
       [g11<=0.5] [g12<=0.09995] [g13<=0.09995] [g16<=0.09995]
        : p=114,np=60,n=0,npq=1,u=28,cx=67,c=1 # 113361
   ...
```

Sample tree of C4.5 (after pruning)

```
g8 <= 9 :
| g11 <= 0.2 :
| | g6 <= 5.4 :
```

```
| | | g10 <= 0 :
| | | | g3 <= 10.6 :
| | | | | g7 > 0 : 0 (92.0/1.4)
| | | | | g7 <= 0 :
| | | | | | g2 <= 19.2 :
| | | | | | | g4 <= 0 :
| | | | | | | | g5 > 0 : 0 (7.0/1.3)
| | | | | | | | g5 <= 0 :
| | | | | | | | | g2 > 4.6 : 1 (51.0/2.6)
| | | | | | | | | g2 <= 4.6 :
| | | | | | | | | | g3 <= 0.2 : 1 (62.0/13.8)
| | | | | | | | | | g3 > 0.2 : 0 (13.0/2.5)
| | | | | | | g4 > 0 :
| | | | | | | | g2 <= 12.2 : 0 (70.0/1.4)
| | | | | | | | g2 > 12.2 :
| | | | | | | | | g5 <= 0 : 1 (6.0/2.3)
| | | | | | | | | g5 > 0 : 0 (18.0/1.3)
| | | | | g2 > 19.2 :
| | | | | | g8 > 0 : 0 (5.0/1.2)
| | | | | | g8 <= 0 :
| | | | | | | g6 <= 0 : 1 (70.0/1.4)
| | | | | | | g6 > 0 :
| | | | | | | | g3 <= 4.2 : 0 (3.0/1.1)
| | | | | | | | g3 > 4.2 : 1 (9.0/1.3)
| | | g3 > 10.6 :
| | | | g11 <= 0 :
| | | | | g9 <= 0 :
| | | | | | g2 <= 51 : 1 (240.0/1.4)
| | | | | | g2 > 51 :
| | | | | | | g6 <= 0.8 : 1 (22.0/1.3)
| | | | | | | g6 > 0.8 : 0 (2.0/1.0)
| | | | | g9 > 0 :
| | | | | | g4 <= 8.4 : 0 (17.0/1.3)
| | | | | | g4 > 8.4 :
| | | | | | | g6 > 2.8 : 1 (33.0/1.4)
| | | | | | | g6 <= 2.8 :
| | | | | | | | g8 <= 0.8 : 1 (15.0/2.5)
| | | | | | | | g8 > 0.8 : 0 (2.0/1.0)
| | | | g11 > 0 :
| | | | | g5 <= 5.2 : 0 (8.0/1.3)
| | | | | g5 > 5.2 :
| | | | | | g1 <= 2.8 : 0 (2.0/1.0)
| | | | | | g1 > 2.8 : 1 (8.0/1.3)
| | g10 > 0 :
| | | g5 <= 7.6 : 0 (205.0/5.0)
| | | g5 > 7.6 :
| | | | g1 <= 3.2 : 0 (25.0/2.5)
| | | | g1 > 3.2 :
| | | | | g10 <= 0.8 : 1 (21.0/1.3)
| | | | | g10 > 0.8 : 0 (2.0/1.0)
| | g6 > 5.4 :
| | | g5 > 9.4 : 1 (268.0/3.8)
| | | g5 <= 9.4 :
| | | | g10 <= 0.8 : 1 (12.0/1.3)
| | | | g10 > 0.8 : 0 (5.0/1.2)
...
```

Sample rules of C4.5rules

```
     g1 <= 6, g6 <= 9, g8 <= 9, g10 > 0.4, g11 > 0.2
->  class 0  [99.6%]
     g5 <= 9.4, g8 <= 9, g10 > 0.8
->  class 0  [99.6%]
     g4 <= 8.4, g6 <= 5.4, g9 > 0
->  class 0  [99.4%]
     g6 <= 5.4, g11 > 0.2
->  class 0  [99.3%]
     g6 <= 11.8, g8 <= 9, g11 > 1.6
->  class 0  [99.3%]
     g3 <= 10.6, g6 <= 5.4, g7 > 0
->  class 0  [99.0%]
     g3 <= 10.6, g6 <= 5.4, g8 > 0
->  class 0  [98.9%]
     g5 <= 5.2, g11 > 0
->  class 0  [98.8%]
     g5 <= 7.6, g8 <= 9, g10 > 0
->  class 0  [98.8%]
     g2 <= 19.2, g3 <= 10.6, g5 > 0, g8 <= 9, g11 <= 0.2
->  class 0  [98.7%]
     g6 <= 2.8, g8 > 0.8, g9 > 0
->  class 0  [98.4%]
     g2 <= 12.2, g3 <= 10.6, g4 > 0, g8 <= 9, g11 <= 0.2
->  class 0  [98.2%]
     g1 <= 1.6, g2 <= 3.2, g8 <= 9, g11 > 1.6
->  class 0  [96.9%]
     g3 <= 4.2, g6 > 0, g11 <= 0.2
->  class 0  [96.1%]
     g1 <= 3.2, g6 <= 5.4, g10 > 0
->  class 0  [95.2%]
     g2 <= 2, g8 <= 9, g11 > 0.2
->  class 0  [93.8%]
     g2 > 51, g6 > 0.8
->  class 0  [92.2%]
     g2 <= 4.6, g3 > 0.2, g3 <= 10.6, g11 <= 0.2
->  class 0  [90.5%]
     g3 <= 2.6, g8 > 9, g9 <= 10.2
->  class 0  [75.8%]
     g2 > 2, g6 > 9, g11 <= 1.6
->  class 1  [99.6%]
     g1 > 3.2, g5 > 7.6, g10 <= 0.8, g11 <= 0.2
->  class 1  [99.6%]
     g2 <= 51, g3 > 10.6, g9 <= 0, g10 <= 0
->  class 1  [99.5%]
     g1 > 6, g6 > 8.2
->  class 1  [99.3%]
     g2 > 19.2, g3 > 4.2, g7 <= 0, g8 <= 0
->  class 1  [99.1%]
     g2 > 3.2, g6 > 11.8
->  class 1  [99.0%]
     g4 > 8.4, g8 <= 0.8, g10 <= 0, g11 <= 0.2
->  class 1  [98.8%]
     g3 > 10.6, g6 <= 0.8, g9 <= 0, g11 <= 0
->  class 1  [98.8%]
     g6 > 5.4, g10 <= 0.4
->  class 1  [98.7%]
     g3 > 2.6, g8 > 9
->  class 1  [98.1%]
     g8 > 9, g9 > 10.2
->  class 1  [97.7%]
     g6 > 2.8, g8 <= 0.4
->  class 1  [96.3%]
     g2 > 12.2, g5 <= 0
->  class 1  [95.8%]
     g2 > 4.6, g4 <= 0, g5 <= 0
->  class 1  [95.5%]
     g3 <= 0.2, g4 <= 0
->  class 1  [84.4%]

Default class: 1
```

Sample rules of CN2 (unordered)

```
IF    g1 < 0.300000, g7 < 20.400002, g8 < 11.100000
THEN  class = "0"  [17 0]
IF    g3 > 48.400002, g9 > 0.300000
THEN  class = "0"  [9 0]
IF    g4 > 5.300000, g7 < 9.500000, g11 > 1.300000
THEN  class = "0"  [252 0]
IF    g1 < 0.700000, g5 < 26.000000, g8 < 8.299999,
      g12 > 0.500000
THEN  class = "0"  [37 0]
IF    g1 < 7.400000, g3 > 34.099998, g9 > 0.700000
THEN  class = "0"  [101 0]
IF    g2 > 43.300003, g8 > 0.300000
THEN  class = "0"  [71 0]
IF    g1 > 56.099998, g5 > 0.500000
THEN  class = "0"  [99 0]
IF    g3 < 2.300000, g5 < 24.599998, g6 > 12.800000,
      g9 > 2.600000, g10 > 1.900000, g11 > 1.200000,
      g12 > 0.300000
THEN  class = "0"  [66 0]
IF    g1 > 77.699997, g2 < 15.900000, g3 > 0.900000,
      g4 > 0.100000
THEN  class = "0"  [74 0]
IF    g2 > 26.100000, g4 < 13.700000, g11 > 0.100000
THEN  class = "0"  [224 0]
IF    g1 > 95.099998, g3 > 0.300000
THEN  class = "0"  [21 0]
IF    g2 < 2.100000, g6 > 26.200001, g10 > 2.300000
THEN  class = "0"  [27 0]
IF    g2 > 10.000000, g3 > 43.300003,
      1.400000 < g6 < 2.900000, g8 > 0.500000
THEN  class = "0"  [20 0]
IF    g1 < 3.100000, g4 > 41.099998
THEN  class = "0"  [7 0]
IF    g4 > 30.299999, g10 > 0.700000
THEN  class = "0"  [62 0]
IF    g2 > 53.500000, g3 < 21.299999, g4 < 4.200000,
      0.900000 < g5 < 2.200000
THEN  class = "0"  [10 0]
IF    56.800003 < g2 < 57.300003
THEN  class = "0"  [1 0]
IF    g3 > 3.300000, g6 > 9.900000, g10 < 8.100000,
      g11 < 4.100000
THEN  class = "1"  [0 334]
IF    g2 > 9.000000, g5 < 0.100000, g6 < 0.100000
THEN  class = "1"  [0 76]
IF    g1 > 2.100000, g4 > 11.900000, g10 < 0.100000,
      g11 < 0.100000
THEN  class = "1"  [0 217]
IF    g3 > 9.900000, g7 < 1.100000, g8 < 0.100000,
      g9 < 0.400000
THEN  class = "1"  [0 181]
...

(DEFAULT) class = "1"  [1070 1090]
```

Sample rules of RIPPER

```
'0' :- g6<=8, g10>=0.2, g10>=0.8 (479/1).
'0' :- g4<=8.4, g7>=0.2, g8<=9, g3<=14.6, g5<=19, g4<=6.8 (201/0).
'0' :- g4<=7.6, g8>=0.2, g8<=8.2, g5<=21.8 (100/6).
'0' :- g1>=75.2, g4>=0.2, g2<=11.8 (69/0).
'0' :- g1<=4.6, g7<=7.2, g11>=0.2, g5<=15, g4<=39.6 (63/0).
'0' :- g5<=9.2, g9>=0.4, g2>=11, g5<=8 (55/1).
'0' :- g3<=2.6, g5>=0.2, g9<=9.4, g4<=8.2 (42/7).
'0' :- g1>=60.8, g6>=0.2 (13/1).
'0' :- g1>=95, g3>=0.2 (15/4).
'0' :- g5<=10.4, g10>=0.2, g2>=30.8 (8/1).
'0' :- g5<=9.8, g2<=7.8, g3>=23.4, g8>=2.8 (6/0).
default '1' (1069/19).
```