# Prognosis of Breast Cancer using Genetic Programming

Simone A. Ludwig and Stefanie Roos

Department of Computer Science, University of Saskatchewan, Canada
ludwig@cs.usask.ca

**Abstract.** *Worldwide, breast cancer is the second most common type of cancer after lung cancer and the fifth most common cause of cancer death. In 2004, breast cancer caused 519,000 deaths worldwide. In order to reduce the cancer deaths and thereby increasing the survival rates an automatic approach is necessary to aid physicians in the prognosis of breast cancer. This paper investigates the prognosis of breast cancer using a machine learning approach, in particular genetic programming, whereas earlier work has approached the prognosis using linear programming. The genetic programming method takes a digitized image of a patient and automatically generates the prediction of the time to recur as well as the disease-free survival time. The breast cancer dataset from the University of California Irvine Machine Learning Repository was used for this study. The evaluation shows that the genetic programming approach outperforms the linear programming approach by 33 %.*

**Keywords:** genetic programming, selection, crossover, mutation, breast cancer, recurrent surface approximation.

## 1 Introduction

Worldwide, breast cancer is the second most common type of cancer after lung cancer (10.4 % of all cancer incidence) and the fifth most common cause of cancer death. In 2004, breast cancer caused 519,000 deaths worldwide (7 % of cancer deaths; almost 1 % of all deaths) [1]. Breast cancer is the most common malignancy in women, except for non-melanoma skin cancers. It continues to be a major health care problem worldwide. Cancer occurs when cells in a part of the body begin to grow out of control. Normal cells divide and grow in an orderly fashion, but cancer cells do not. They continue to grow and crowd out normal cells. Although there are many kinds of cancer, they all have in common this out-of-control growth of cells [2].

Different kinds of cancer can behave very differently. For example, lung cancer and breast cancer are very different diseases. They grow at different rates and respond to different treatments. That is why people with cancer need treatment that is aimed at their kind of cancer. Therefore, it is important to identify the type of cancer accurately, so that the correct treatment can be started.

Breast cancer is a cancer that starts in the tissues of the breast. There are two main types of breast cancer [3]: (1) Ductal carcinoma starts in the tubes (ducts) that move milk from the breast to the nipple. Most breast cancers are of this type. (2) Lobular carcinoma starts in parts of the breast, called lobules that produce milk.

The good news is that early detection and new treatments have improved survival rates of breast cancer. The 5-year survival rate for women diagnosed with cancer is 80 %. About 88 % of women diagnosed with breast cancer will survive at least 10 years. Unfortunately, women in lower social and economic groups still have significantly lower survival rates than women in higher groups. The good news is that women are living longer with breast cancer. Survivors must live with the uncertainties of possible recurrent cancer and some risk for complications from the treatment itself [4].

Recurrences of cancer usually develop within 5 years of treatment. However, 25 % of recurrences and half of new cancers in the opposite breast occur after 5 years.

In order to aid the physicians in the prognosis of whether the breast cancer is likely to recur in patients, the linear programming approach has be introduced in 1994 [7]. It has achieved good results with an expected error of 13.0 to 18.3 months, which was better than the prognosis correctness achieved by other available techniques at that time. However, the proposed approach outlined in this paper, takes another step towards the improvement of the prognosis correctness using a genetic programming approach.

This paper is structured as follows. First, some background information regarding the previous work using the linear programming technique, the data collection and analysis is outlined in Section 2. In Section 3, the proposed approach using genetic programming is introduced outlining the different parameters involved. The experiments and results are given in Section 4. Section 5 concludes this paper with a summary and analysis of the results obtained.

## 2 Previous Work

The dataset used for the prognosis of breast cancer is publicly available at the UCI Machine Learning Repository [5]. The dataset was collected by researchers from the University of Wisconsin as follows. First, a sample of fluid was taken from the patient's breast. This fluid was then placed on a glass slide and stained to highlight the nuclei of the constituent cells. An image from the FNA is transferred to a workstation by a video camera mounted on a microscope. A program, called Xcyt [6], was developed which uses a curve-fitting program to determine the exact boundaries of the nuclei. The boundaries were initialized by an operator using a mouse pointer. Ten features were computed for each nucleus: area, radius, perimeter, symmetry, number and size of concavities, fractal dimension (of the boundary), compactness, smoothness (local variation of radial segments), and texture (variance of gray levels inside the boundary). The mean value, extreme value (i.e. largest or worst value: biggest size, most irregular shape) and standard error of each of these cellular features were computed for each image. Furthermore, tumor size and lymph node status make up a total of 32 real-valued features. The dataset contains 198 observations; with 47 recur cases and 151 non-recur cases.

Using this dataset, Street et al. [7] applied linear programming to predict the Time To Recur (TTR), which is a mapping of an n-dimensional input of cytological and other features to a one-dimensional time output. One complicating factor for the prediction is that TTR is known for only a subset of patients and not for others for which we know only the time of their last check-up, or Disease-Free Survival time (DFS). However, all available cases were used for this investigation. The solution to this estimation program is termed the Recurrence Surface Approximation (RSA) technique [7,8,9]. RSA basically uses linear programming to determine a linear combination of the input features that accurately predicts TTR. The linear program to be solved for a given training set is given in [8]. The motivation for the RSA approach was the following: (1) Recurrences actually take place at some point in time prior to their detection. However, the difference between the time a recurrence is detectable (actual TTR) and the time it is actually detected (observed TTR) is assumed to be small. (2) Observed DFS is a lower bound on the recurrence time of that patient.

Therefore, three types of absolute errors need to be considered. Let $err_1$ be the average overestimation using only the set of patients with a recurrence (i.e., the set of recurrent cases). The average underestimation of DFS using only the set of patients without an observed recurrence is denoted with $err_2$. Since DFS is a lower bound for the TTR, overestimation is not considered an error. Thus, the third type of error, $err_3$, is the average underestimation of TTR (on the set of patients having a recurrence). Underestimation of TTR is not considered as severe as overestimation, hence the RSA approach tries to minimize: $err_1 + err_2 + \delta \cdot err_3$ (1)
where $0 < \delta \leq 1$ defines the influence of underestimating TTR in relation to overestimating TTR. For the genetic programming approach, Equation (1) is used as the fitness function, thus allowing a direct comparison. Note that it is not sensible for machine learning algorithms to set $\delta = 0$, even considering that the recurrence appeared at some point before it was detected. Allowing $\delta = 0$, the algorithm could simply perform a classification into recurrent and non-recurrent cases. Then 0 could be predicted for all recurrent cases (underestimating TTR in all cases), while some large value (meaning greater than the maximal DFS in the set) is predicted for the non-recurrent cases (always overestimating DFS). However, the goal of this research is to perform regression analysis for medical prognostics, and therefore, symbolic regression and not classification was used.

In [7], $\delta$ was chosen such, that of those values for which $err_1 + err_2$ are minimal, the one that minimizes $err_3$ is chosen. Based on the perturbation theorem [10], such that a $\delta,\ 0 < \delta \leq \bar{\delta}$ for some $\bar{\delta}$, exists. For the proposed genetic programming approach, the influence of various δ on the fitness and the expected error is investigated, as well as on $err_1$, $err_2$ and $err_3$ each.

In order to obtain the best generalization, it is important to choose the right subset of features. The appropriate feature set was chosen for the linear programming approach in the following automatic fashion. A tuning test (one tenth of the training cases) was first set aside. The RSA linear program was then solved using all input features, and the resulting surface was tested on the tuning set. Features were then removed one by one. Each new problem was solved and the result tested on the tuning set, until only one feature remained. Using the features that showed the best performance on the tuning set, all the training data was then re-optimized. Feature selection is done differently for the proposed genetic algorithm approach and will be outlined in the following section.

## 3 Genetic Programming Approach

The origins of evolutionary computation reach back to the 50's of the last century. Genetic programming, in itself, was not considered until the middle of the 80's. The term first appeared in [11], the main development took place in the early and middle 90's, particularly through work by Koza [12].

Genetic programming uses the concepts of genetics and Darwinian natural selection to generate and evolve entire computer programs. Genetic programming largely resembles genetic algorithms in terms of its basic algorithm. The notions of mutation, reproduction (crossover) and fitness are essentially the same, however, genetic programming requires special attention when using those operations. While genetic algorithms are concerned with modifying fixed-length strings, usually associated with parameters to a function, genetic programming is concerned with actually creating and manipulating the (non-fixed length) structure of the program (or function). Therefore, genetic programming is more complex than genetic algorithms [13] and works as follows. In genetic programming the aim is to find solutions to some problem in the form of a computer program. It is a stochastic search strategy that is particularly powerful in the circumstances where one cannot make any assumptions about the characteristics of the solution. The solution is developed by first creating a number of initial programs, which are then recombined and changed in each evolution step [14].

Genetic programming performs the following steps:

```
Step 1: Assign the maximum number of generations to be run and probabilities for cloning, crossover and mutation.
Step 2: Generate an initial population of computer programs of size N by combining randomly selected functions and terminals.
Step 3: Execute each computer program in the population and calculate its fitness with an appropriate fitness function. Designate the best-so-far individual as the result of the run.
Step 4: With the assigned probabilities, select a genetic operator to perform cloning, crossover or mutation.
Step 5: If cloning operator is chosen, select one computer program from the current population of programs and copy it into a new population. If crossover operator is chosen, select a pair of computer programs from the current population, create a pair of offspring programs and place them into the new population. If mutation operator is chosen, select one computer program from the current population, perform mutation and place the mutant into the new population.
Step 6: Repeat Step 4 until the size of the new population of computer programs becomes equal to the size of the initial population N.
Step 7: Replace the current (parent) population with the new (offspring) population.
Step 8: Go to Step 3 and repeat the process until the termination criterion is satisfied.
```

The Java Genetic Algorithms Package (JGAP) [15] was chosen as the programming platform. JGAP is a Genetic Algorithms and Genetic Programming package written in Java. It is designed to require minimum effort to use, but is also designed to be highly modular. It provides basic genetic mechanisms that can be used to apply evolutionary principles to solve problems.

## 4 Experiments and Results

The experiments were done as follows. First of all, the dataset was pre-processed normalizing the values by subtracting the mean and dividing by the standard deviation. There were 4 missing values for the lymph node status, which were replaced by the mean. For the division of training and test data, the leave-one-out method was used as this sampling method was also used for the linear programming approach. Then, the genetic programming was fine-tuned with the selection of parameters such as function set, feature selection, maximal crossover depth, crossover and mutation rate, population size, and number of generations. Afterwards, the varying δ values were investigated.

In general for the fine-tuning, the initial choice of parameters was: tournament selection of size 4, population size 1,000, minimal initial depth 5, maximal crossover depth 12, crossover rate 0.9, mutation rate 0.3, and a δ value of 0.01. Basic mathematical operations (addition, subtraction, multiplication, division), comparison operators (<,>), if-statement, logarithm and exponential made up the function set. For those parameters that were analysed, the best performing value was used for all following tests. If not otherwise stated, 30 tests were performed for each choice of parameter. Note that the results were compared using the average expected error, not the fitness value. However, the lowest expected error always corresponds to the lowest fitness value. The expected error only considers known errors, meaning the underestimation of DFS and the overestimation of TTR.

*Feature selection:* 180 test cases over 300 generations were run, and it was counted how often each feature showed up in the best solution of the run. Afterwards test cases each with the 4 and 8 most frequent features (all others were contained in less than 10 %) were run. The best result was achieved when 8 features were used: 13.02 in comparison to 13.62 (4 features). Using all features led to an average error of 13.80. The 8 features are (in the order of the number of times they are used): Tumor size, lymph node status, mean symmetry, extreme area, standard deviation of radius, standard deviation of area, extreme compactness, and extreme concave points. This is quite different to the 5 features found by Street et al. [5]: mean area, mean perimeter, mean fractal dimension, extreme value for area and perimeter. The explanation for this lies in the resulting programs of the genetic programming method which produces, given the function set used, also if conditions. An example of part of the resulting programs frequently contained the condition: `if tumor size > mean symmetry and lymph node status > 0.0` … . Such relationships between the different features can influence the outcome more than the actual values of the features.

*Function set:* 300 generations were used and every function set tested contained at least the four basic mathematical operations (addition, subtraction, multiplication and division). Additionally, function sets using comparison operators, if-statements, logarithm, exponential, min, max, cosine, sine, random numbers or sigmoid functions were considered. The best performance was achieved when using nothing but basic mathematical operations, comparison operators and if-statements. Function sets without the later two performed considerably worse (above 15), while function sets consisting at least of the above functions had on average an expected error of 12.60 to 14.44.

For reasons of time constraints only 200 generations were used for finding the next 3 parameters (maximal crossover depth, crossover and mutation rates).

*Maximal crossover depth:* Maximal crossover depths of 9, 12 and 17 were tested. Using 9 resulted in a higher expected error (13.61) than the 12 or 17.

*Crossover and mutation:* The crossover rates 0.5, 0.7, 0.8, 0.9 were used, together with mutation rates between 0.1 and 0.5. The crossover rate of 0.9 and the mutation rate of 0.1 produced the highest expected error of 13.09.

*Population size:* Taking 2,000 individuals and 300 generations improved the average error to 12.24. For 5,000 and 10,000 individuals only 10 test cases were run. In case of 5,000 individuals the expected error decreases significantly to 10.88. Surprisingly, the test runs with a population size of 10,000 do not achieve a similar increased performance. With an average expected error of 12.17, the performance is hardly better than for a population size of 2,000. This might be due to the low number of tests, so that one or two tests with a bad result influence the average result significantly. More tests would be needed to verify if 5,000 is indeed the best choice for the population size. However, using higher population sizes increases the time cost. The execution times measured, taking the average of 10 runs, are shown in Figure 1. As estimated, a linear increase in the execution time can be observed while increasing the population size.

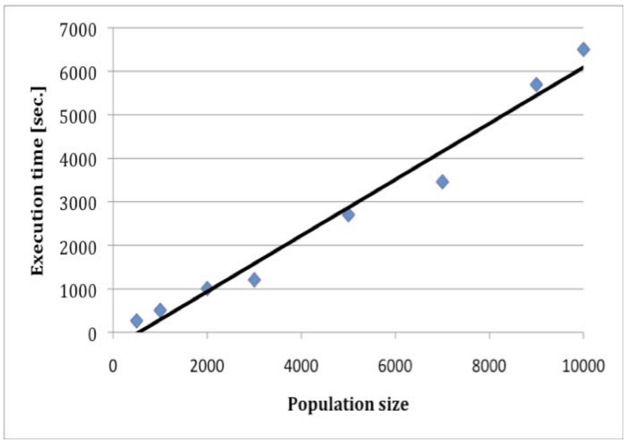

**Fig. 1.** Performance vs. population size.

*Number of generations:* Running 10 tests with 1,000 generations resulted in an expected error of 10.9. On average, the best result was found after 734 generations.

The second part of the evaluation was to investigate the average error in comparison to varying $\delta$ values. The best parameter settings found earlier were used: population size: 2,000, number of generations: 300, mutation rate: 0.1, crossover rate: 0.9, maximal init depth: 5, maximal crossover depth: 9, new individuals per generation: 0.1, function set: addition, subtraction, multiplication, division, logical and, logical or, >, <, if, feature set: tumor size, lymph node status, mean symmetry, extreme radius, standard deviation of radius, extreme perimeter, mean fractal dimension and extreme area of cell nuclei.

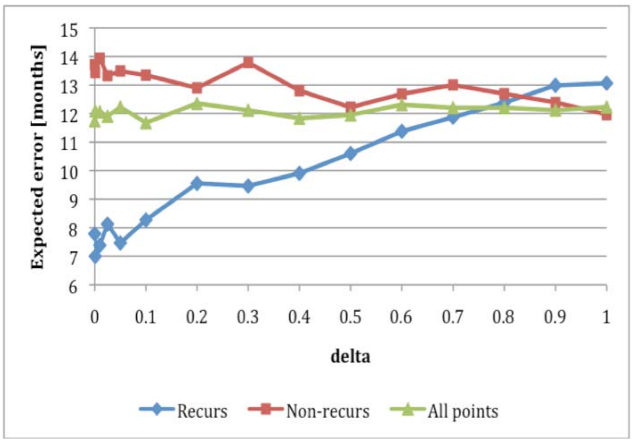

**Fig. 2.** Expected error vs. varying $\delta$ values.

Figure 2 shows the expected error for varying $\delta$ values. As can be seen, the error for the recurs is clearly increasing with the value of δ, corresponding to the higher emphasis on underestimation of TTR in contrast to overestimation. The increase, considering all data points, is less noticeable since the slight decrease of the expected error on the non-recurs balances the increase on the recurs to some extent. The performance on recurs is better for low values of δ, while for higher values the performance on the non-recurs exceeds the one on recurs. From this one might guess that the prognosis of TTR works better than the one of DFS. After all, for δ=0 the underestimation of DFS and the overestimation of TTR are given the same weight in the fitness function, thus one might expect to get a similar performance. For δ=1, on the other hand, underestimation contributes to the fitness value for all 198 data points, but overestimation only for the 47 recur cases.

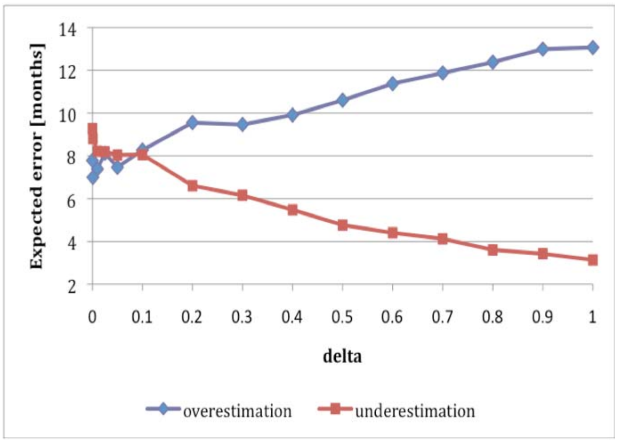

**Fig. 3.** Expected error of recur values for overestimation and underestimation.

Figure 3 shows the overestimation and underestimation of the recur values. As mentioned before, δ determines the relation between the overestimation and the underestimation of TTR. It can be clearly seen that the programs are not able to focus very well on preventing overestimation when δ is higher, since they have to control underestimation of recurs as well as to minimize the fitness value. This leads to a higher emphasis on controlling underestimation on the whole dataset, explaining the slight decrease of the expected error on non-recurs with increasing δ. Note that preventing underestimation of TTR seems to work better than preventing overestimation. For high values of δ, the average underestimation is only 3.1 months, while the best result for the overestimation is 7.4 months, achieved with δ=0.01.

In Figure 4, the fitness value for Equation (1) including the $\delta$ term and without the $\delta$ term is shown. As can be seen, the larger the $\delta$ value gets, the larger the difference between the two curves gets, indicating the increase in importance of the third term. As this term becomes more important for the fitness, the programs focus more on the third term than on the remaining two. Therefore, we can see a higher fitness value without the delta term.

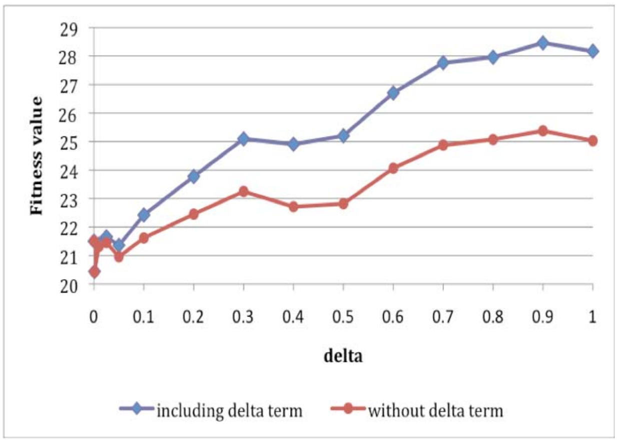

**Fig. 4.** Investigation of $\delta$ term in fitness function.

Taking Figure 2 and 3 into account, the higher fitness value can be explained by the increased expected error for the recur cases. In contrast to measuring the performance on all data points, the fitness measure places an equal weight on the two classes. This is why the fitness value is influenced more significantly by a higher error on recurs than the overall performance, which puts an equal weight on each data point, thus the error on the non-recurs (151 cases) is more important than the error on the 47 recur cases.

The best values achieved with the genetic programming method compared to the linear programming technique for $\delta = 0.1$, shows that genetic programming achieves a higher accuracy on all prognostic formulations, as shown in Table 1; the improvement is above 33 %.

**Table 1.** Comparison of Linear Programming (LP) with Genetic Programming (GP) Results.

| | All points | Non-recur | Recur |
|---|---|---|---|
| **LP** | 18.3 months | 19.9 months | 13.0 months |
| **GP** | 11.7 months | 13.3 months | 8.3 months |
| **Improvement** | 36.1 % | 33.2 % | 36.2 % |

## 4 Conclusion

Breast cancer victims' chances for long-term survival are improved by early detection of the disease. Early detection in turn is enhanced by an accurate diagnosis. The choice of appropriate treatments immediately following surgery is largely influenced by prognosis, which provides the expected long-term behaviour of the disease. Therefore, an automatic method to aid physicians in their diagnosis and prognosis is of essence.

Previous work regarding the prognosis of breast cancer used a linear programming approach, which arrived at an expected error of 13.0 to 18.3 months, which at that time, attained a better prognosis correctness than other available techniques.

The proposed approach used genetic programming to enhance the prognosis accuracy even further. The fine-tuning of feature selection, selection of the function set, maximal initial depth and crossover depth, crossover and mutation, population size and the number of generations achieved an expected error between 11.7 and 12.4 months, which accounts for an above 36 % higher prognosis accuracy. The main contributor for the higher accuracy was the selection of functions that increased the accuracy. Given that the feature selection of both techniques resulted in a different feature set implies that feature selection is specific to the learning technique. In particular, the possibility of genetic programming to include non-linear functions and if-statements seems to make the difference. The performance for a function set without if-statements is only slightly better than the results achieved by Street et al., which resulted in an expected error of 15.63.

Future work involves running the genetic programming approach with a larger population size, as further improvement can be achieved. In addition, the number of generations also increases the accuracy, however, again at the cost of the execution time. As a larger population size and a larger number of generations imply a longer execution time, the parallelization of the genetic programming could achieve a better performance.

**Acknowledgments.** The authors would like to thank the Natural Sciences and Engineering Research Council (NSERC) for the partial funding of this research.

## References

1. World Health Organization, "Fact sheet No. 297: Cancer", http://www.who.int/mediacentre/factsheets/fs297/en/index.html, last retrieved October 2009.

2. K. Nouri, “Skin Cancer”, McGraw-Hill, 2007.
3. Calgary Breast Health Program, http://www.calgaryhealthregion.ca/breasthealth/breast_cancer/types_of_breast_cancer.htm, last retrieved May 2010.
4. Breast Cancer Prognosis, http://www.healthcentral.com/breast-cancer/, last retrieved May 2010.
5. Prognostic Wisconsin Breast Cancer Database, UCI Machine Learning Repository, http://www.ics.uci.edu/~mlearn/MLRepository.html, last retrieved May 2010.
6. W. N. Street, “Cancer Diagnosis and Prognosis via Linear-Programming-Based Machine Learning”, PhD thesis, University of Wisconsin-Madison, August 1994.
7. W. N. Street, O. L. Mangasarian and W. H. Wolberg, “An Inductive Learning Approach to Prognostic Prediction”, Proceedings of the Twelfth International Conference on Machine Learning, pp. 522-530, Morgan Kaufmann, 1995.
8. O. L. Mangasarian, W. N. Street, and W. H. Wolberg, “Breast cancer diagnosis and prognosis via linear programming”, Journal of Operations Research, vol. 43, pp. 570-577, 1995.
9. W. H. Wolberg, W. N. Street, D. H. Heisey, and O. L. Mangasarian, “Computer-derived nuclear grade and breast cancer prognosis”, Journal of Analytical and Quantitative Cytology and Histology, vol. 17, no. 4, pp. 257-264, 1995.
10. O. L. Mangasarian and R. R. Meyer, “Nonlinear perturbation of linear programs”, SIAM Journal on Control and Optimization, vol. 17, pp. 745-752, 1979.
11. N. L. Cramer, “A representation for the Adaptive Generation of Simple Sequential Programs”, Proceedings of an International Conference on Genetic Algorithms and the Applications, Grefenstette, 1985.
12. J. R. Koza, “Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, 1992.
13. W. Banzhaf, P. Nordin, R. E. Keller, F. D. Francone, “Genetic programming: an introduction on the automatic evolution of computer programs and its applications”, Morgan Kaufmann Publishers, 1998.
14. R. Poli, W. B. Langdon, N. F. McPhee. “A Field Guide to Genetic Programming”, http://www.gp-eld-guide.org.uk, last retrieved May 2010.
15. Java Genetic Algorithms Package (JGAP), source code, http://jgap.sourceforge.net, last May 2010.