Using Genetic Algorithms in Software Optimization

ION IVAN, CATALIN BOJA, MARIUS VOCHIN, IULIAN NITESCU, CRISTIAN TOMA, MARIUS POPA
Economic Informatics Department
Academy of Economic Studies
Romana Square No. 6, Bucharest
ROMANIA

Abstract: Concepts of informatics application and software optimization are defined. Presented are criteria and graphical methods for optimization. As far as multi-criteria optimization for very complex software applications is concerned, genetic algorithms are proposed. Their effects are compared with classic algorithm effects for the same lot of applications.

Key-Words: optimization, software, genetic algorithms, compiler

1. Software applications

Software applications represent complex structures designed to solve precisely defined problems. The software application implies a software component for processing, input and output data which is the result of processing. It is a triple defined by input data, software and results. It is considered that the software application exists if and only if for complete and correct input data, after the processing stage complete and correct results are obtained which fully satisfy the user's requirements.
With an increase in program complexity different management strategies are implemented. These administrate the product's quality or the quantity of resources needed. The management strategies are also involved in the lifecycle stages of the final product and are confronted with a large number of primary factors and aggregated factors, whose levels have to be noted. The situation imposes the software product to be modularized into a number of different specialized components. This presents the following advantages:

- reusability of the modules, including them in other products;
- a better administration of the analysis, development and implementation phases;
- a clearer view of the functional components;
- specialization of the analysts, designers and programmers.

There are numerous criteria for classifying software applications. Based on the structure criteria, software applications are:

- software applications with a linear structure;
- software applications with a tree structure;
- software applications with a network structure.

Based on the type of application:

- applications; these implement data processing operations and contain functions from various socio-economical fields;
- multimedia; these give the necessary support for processing video and audio files;
- operating systems; these administrate the computers' hardware resources and permit access to them for software applications;
- games; represent applications designed for entertainment or user education;
- utilities and instruments; are used for administering and protecting different functions of the operating system or other software applications.

The society based upon knowledge is characterized by using software applications included in architectures of large applicability.

2. Optimum criteria

Software quality is a complex concept used to describe a software product judged by the economic-technical characteristics. On the basis of these characteristics a software application can be identified as being better or worse than another one from the same category. The process of quality analysis is based on a complex system of quality characteristics. The quality factors taken into account are:

- efficiency through the reduction time and resources;
- functionality through correctness, security, compatibility, completeness and interoperability;
- maintainability which permits correcting deficiencies and future development of the product;

- portability which ensures independence from hardware, independence from software platforms and reusing the application in different software products;
- robustness through tolerance to errors and through the extent to which the product is operational during hardware or software failure.
- usability which describes the effort made by users to understand the workings of the product.

The optimum criteria are defined on the basis of the above. The criteria become objectives in the optimization process because they describe both quantitatively and qualitatively what is the goal in the development cycle of the software product.

Minimizing the workload. The workload decisively influences the processing duration because it is defined as the number of clock cycles, or the number of instructions executed. The clock cycle or the executed instruction is characterized by durations, and an increased number of clock cycles or executed instructions lead to an increase in program execution duration. To optimize a program means to make such a construction that will lead to getting the final results in the shortest possible time. Thus, to optimize a program means to find ways in which to reduce the workload as much as possible, i.e. to minimize it.

Maximizing precision. Precision is a new concept with regards to the limitations that are imposed by existing representations in computers. We work at byte level, at word level (two bytes), at double word level, at double precision and with 10 byte memory zones. For every type a domain and representation are defined as well as stating the effects different mathematical operations have with different operands defined by these types, upon condition indicators. The programmer has to carry out studies regarding the operands domains to make sure the results are guaranteed to be correct and do not generate uncontrolled interrupts in the case of distributes applications.

Maximizing the dimension of the problem to be solved. Any problem is characterised by the input data stored in some data structure, the algorithm steps needed to be taken to process the data, the final result and maintenance requirements for the informatics solution chosen.

Maximizing the dimension of the problem means creating a software application that includes a maximised set of functions tied to the objective for which the software application was developed for. The criterion sets the degree of generalisation of the problem by including considered factors of influence, hypothesis for calculating and methods for solving the problem. Maximizing the dimension of the problem is a trend which the programmer has to follow simultaneously with the duration of solving the problem the software was written for.

Maximizing the generalisation of the software product. Any problem put forward for elaborating software analysis has to be undertaken gradually. A simpler form is taken, after which further developments appear. During this time the programmer gains the necessary experience for correctly approaching the problem and understanding the efforts needed to move from a complexity level to a raised level of complexity for the problem.

If at first, the problem takes on the most complex form, with the highest degree of generalisation, there are risks of it not being understood, and the beneficiaries could wait a long time before the first correct results come in. Usually the costs of such an approach are a lot greater than if the problem was approached in a gradual way, by going through different stages. This means going from a from that solves a simpler problem to a program that solves a more complex problem, therefore building the program a step at a time.

3. Classical optimizations for software applications

The evolution of software and hardware technologies permits for complex software applications in the present, but also with great requirements for processing speed and memory usage. Software applications included in this category are operating systems, entertainment applications and multimedia applications.

With all this waste of resources, transparent to the user, the developer gives particular importance to the optimization process, looking to maximize the performance of the final product.

Another category of software products is constrained from the start to be efficient with respect to the system resources used. In this category are included antivirus applications, drivers, viruses, applications implemented in microcontrollers or smartcards, function libraries, applications for mobile devices.

The objective of software optimization is to obtain a new product or a new version of an existing product, which presents a higher quality level. This grade is worked out based on the levels obtained from the set of software characteristics or the established optimum criteria. These are well defined and are followed throughout the whole process, directly measuring the obtained levels. By direct comparison to the base levels or by determining aggregated values based on the way multi-criteria models are composed, the level of improvement is obtained. One such model, [10], permits determining an aggregated quality level which measures the effects of multi-criteria optimization.

The optimization process implements techniques and methods used in:

- *problem analysis*; this implies that a lot of the problems in software optimization are generated in stages before the development of the source code; the implementation of an inefficient analysis leads to defining a solution that isn't characterised by a required quality level;
- *source code*; if this is based on a bad implementation of an algorithm, it will lead to obtaining inefficient results in most situations, even if the complexity of the source code may be reduced, or if it is of high quality; the main cause of problems at source code level lies in a low level of its experience, and last but not least in the mistakes it makes; the primary methods, [10], used at this level are based on the elimination of repeating sub-expressions, instructions without any meaning, sequences in which instructions with opposite effect appear, invariations, by substituting complex reference expressions with simpler ones and by regrouping control structures;
- *compiler*; this component is responsible for transforming source code in the form associated with the high level language into machine code; as this form is directly processed by the microprocessor, it greatly influences the way in which resources are used and especially the total processing time; using a good complier that contains a lot of techniques for optimizing memory usage and processing speed leads to optimizing the application without the need for any other additional effort; the second solution for getting optimized machine code is to write the source code in assembler languages; analysing the efficiency of using an optimization routine for the compiler or writing the application in assembler languages leads to defining two approaches; in the case of routines with a high level of importance for application performance, the programmer can generate machine code more efficiently than the compiler; the solution of developing the whole application in machine code is not viable because the effort and time resources are too great;

4. Genetic algorithms in optimization

Optimizing compiler options represents one of the areas in which genetic algorithms show their capability. Genetic algorithms search for solutions to problems by imitating mechanisms specific to natural evolution. To find a solution, a population is constructed, each individual representing a possible solution to the problem. Transformations inspired from natural evolution are applied upon the population: selection, crossover, mutation. In the evolution process the individuals become more and more adapted to the environment [4].

The *gcc* compiler presents hundreds of options: for code optimization, for pre-processing, for link-editing, options specific for C++ or C, dependant on the computer etc. For finding the optimum combination all the possible combinations have to be tried out, but the search space is too vast. For determining the optimal solution in a reasonable time, genetic algorithms are proposed. The chromosome is an ordered set of elements, named genes, whose values determine the characteristics of an individual from the population. The chromosome is represented by arrays which contain the coding for a possible solution.

The configuration options are held in a list. The chromosome is represented like a string of bits. If the element at position i from the string has value 1, then the option from position i in the array is used to compile the program.

If the compiler options array and the chromosome are set as follows:

Table 1. Compiler options and chromosomes.

Position	1	2	3	4	…	n-1	N
Options	-lrt	-lm	-01	-fgcse	…	-fno math-errno	-mieee-fp
Chromosome	**1**	**1**	**1**	**0**		**0**	**0**

Then the configuration options are –lrt, -lm and -01.

The value for the *fitness* function for an individual is given as the execution time of the program compiled with the options chosen based on the certain chromosome.

The initial population is randomly generated, and is made up of a relatively small number of individuals. The generated chromosomes have to be checked if they are valid. Tests have to be made to make sure if compulsory options have been included and if there are options that are in conflict with one another.

Chromosome selection for reproduction is done using the roulette method and the elitist strategy. The roulette method permits reducing the selection pressure, maintaining population diversity. The elitist strategy refers to keeping the best chromosomes, for which the value of the fitness function is the greatest, from one generation to the next.

When choosing the elitist strategy, it has been taken into account that from one generation to the other the performance criterion does not change. The strategy makes sure that each generation contributes in a positive manner to the execution speed of the program. This way the algorithm converges faster to an optimal solution [7]. The method assures a high rate of convergence, but to avoid premature convergence the method has to be used with mechanisms for maintaining the population diversity.

Crossover with a single cut-off point will be used. Two parents are chosen (x and y) which generate two descendants (x' and y'). A k between 1 and n-1 is chosen randomly, named a cut-off point, and the descendants are constructed:

$$x'=(x_1,\dots,x_k,y_{k+1},\dots,y_n) \text{ şi } y'=(y_1,\dots y_k,x_{k+1},\dots,x_n) \quad (1)$$

The mutation assures the alternation of gene values, making sure population diversity is kept. Mutation takes place with a very low probability, being a secondary operator for genetic algorithms. A gene is chosen randomly and its value is changed.

After applying the crossover and mutation operators the validity of each chromosome is tested. A chromosome is invalid if it contains non-permitted combinations of compiler options or if it doesn't contain compulsory options. For verification and correction correspondence matrixes are used, and the genes are set accordingly.

If the population is small, the genetic algorithm has a better execution speed, but searching in the solution space is inefficient. If the population is large, its accuracy increases, but the speed decreases. Because if this, it is advised that it is used together with other methods. Genetic algorithms generate a small number of populations, and the best result is used as an entry point for classical search methods.

Optimizing the distribution of data destined for network processing uses genetic algorithms because the volume of data is diverse and its dynamics is particular. When a distributed database system is designed, an important requirement is that it uses the network in an optimal way. Each node has to execute on average the same number of queries. At the same time, a query has to access as less nodes as possible. By equilibrating the two requirements, an optimal distributes database system is obtained. Genetic algorithms are used to find the solution.

Consider a table with 16 entries, with two columns: car name and its colour. The accepted colours are: blue (A), yellow (G), red (R), green (V). The number of blue cars is 5, yellow 4, red 3, green 4. This table will be distributes in a network with 2 nodes. Three possible configurations are possible:

Table 2. Data distribution in nodes.

Node 1	Node 2
A A A G R R V V	A A G G G R V V
A A G G R V V V	A A A G G R R V
A G G G R R V V	A A A A G R V V

Consider a table with n entries, and network with k nodes. The chromosome is coded as an array with n elements, each having a value between 1 and k-1. If element i from the vector has value x then entry i from the table is saved in node x. The chromosome has a length equal to the number of entries.

Chromosome selection for reproduction is done using the roulette method and the elitist strategy.

The *fitness* function has to include the following optimum criteria: grouping of entries with the same attributes, as well as distributing them in an equal fashion between a minimum number of nodes.

Grouping entries with the same attributes is done using the following formula [3]:

$$GI = \sum_{i=1}^{k} \sum_{j=1}^{m} \left(\frac{A(i,j)}{n_i} \right)^2 \quad (2)$$

where:

GI	– attribute groping
k	– number of distinct attributes
m	– number of nodes
A(i,j)	– number of attributes i from node j
n_i	– total number of attributes i

Distribution of entries with the same attributes between a minimum number of nodes if done using the following formula:

$$DI = \sum_{i=1}^{k} \sum_{j=1}^{m} |A(i,j) - b| \quad (3)$$

where:

DI	– distribution of entries
b	– maximum number of entries that can be held in a node

The fitness function is:

$$F = coefa\frac{GI}{k} + coefb\frac{1}{1+DI}, coefa + coefb = 1,\ coefa\, si\, coefb < 1$$

(4)

Where *coefa*, *coefb* are 2 constants; *coefa*, *coefb* are chosen based on the importance given to each optimum criterion.

The crossover is done through random inheritance common to both parents.

Table 3. Data distribution in nodes.

Entries	1 2 3 4 5 6 7 8 9 10 11 12
Colours	A A R G G R R G A V V R
Parent 1	1 2 1 2 2 4 1 4 2 3 3 1
Parent 2	2 2 1 1 2 3 1 4 2 2 3 1
Child	x 2 1 x 2 x 1 4 2 x 3 1

Each node I is allocated a maximum number of $\frac{n}{k}+1$ entries, where n represents the number of entries from the table and k the number of nodes. Taking into account the long length of the chromosome, the completion algorithm has to be as simple as possible for a fast execution and for reducing memory usage. The completion algorithm has to group for each node entries with the same attribute.
For entry 1, node 2 is chosen because this already contains 2 entries with the same attribute. For entry 4 node 2 or 4 is chosen randomly. The mutation probability is decided at gene level. A number r is randomly chosen between 1 and n-1. A gene is chosen and the mutation operator is applied with a very small probability. Its value is interchanged with the value from another gene from the chromosome. The mutation algorithm is repeated for r genes.

5. Comparative analysis of the effects of genetic algorithm optimization

Testing the program for optimising compiler options permits tracking the effects of processes developed through genetic algorithms. For the testing, 4 programs were written in C++. The programs are compiled using g++, under OpenSuse 10.2, with the compiler options offered by the optimizing program.
Settings for the genetic algorithm are: population number = 60, number of selected individuals = 40, number of generations = 100, mutation probability = 0.04.
The first program instantiates 400000 object dynamically allocated, and then immediately disposes them.

Table 4. The best 3 results from the first program

Compiler options	Execution time (ms)
-O1 -fno-guess-branch-probability -fno-cprop-registers -fno-thread-jumps -fno-if-conversion -ftree-ccp -fprefetch-loop-arrays -fpeel-loops -fbranch-target-load-optimize	182
-lrt -lm -O1 -fno-merge-constants -fno-cprop-registers -fno-thread-jumps -fno-if-conversion -fno-loop-optimize -ftree-dce -ffloat-store -fno-inline -fpeel-loops -funswitch-loops	216
-std=gnu99 -O1 -fno-cprop-registers -fno-if-conversion -fno-delayed-branch -fno-loop-optimize -ftree-ccp -fprefetch-loop-arrays -fpeel-loops -fbranch-target-load-optimize	275

The second program does the linpack test which determines the performance of a processor.

Table 5. The best 3 results from the second program

Compiler options	Execution time (ms)
-lm –std=gnu99 -O1 -fno-merge-constants –fno-defer-pop -fno-cprop-registers -fno-if-conversion -fno-loop-optimize -ftree-ccp -funswitch-loops	418
-lm –std=gnu99 -O1 -fno-merge-constants -fno-defer-pop -fno-thread-jumps -ftree-ccp -ftree-dce -funswitch-loops	440
-lrt -lm -std=gnu99 -O1 -fno-defer-pop -fno-cprop-registers -fno-thread-jumps -ftree-dce	788

In figure 1 you can see the dynamics of the network learning process and its effects upon the results.

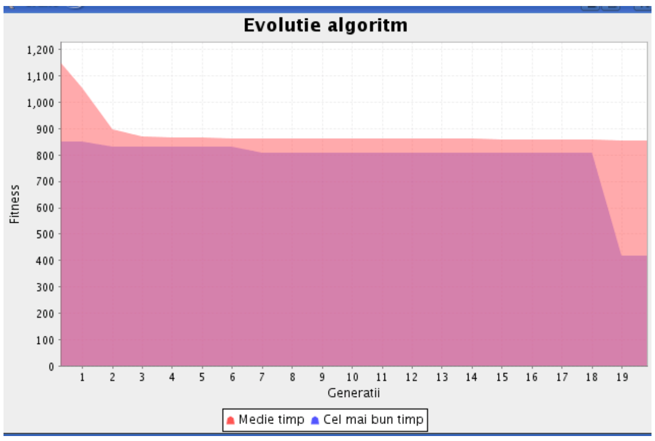

Fig. 1 Evolution of the genetic algorithm for the second program.

The third program multiplies 2500 times two matrixes of dimension 30x30 dynamically allocated.

Table 6. The best 3 results from the third program

Compiler options	Execution

	time (ms)
-std=gnu99 -O1 -fno-if-conversion -ftree-dce -ffloat-store	723
-lm -std=gnu99 -O1 -fno-merge-constants -fno-defer-pop -fno-thread-jumps -fno-if-conversion -fno-delayed-branch -fno-loop-optimize -fprefetch-loop-arrays -fpeel-loops -funswitch-loops -fbranch-target-load-optimize	782
-lm -std=gnu99 -O1 -fno-merge-constants -fno-cprop-registers -fno-thread-jumps -fno-if-conversion -fno-delayed-branch -fno-loop-optimize -ffloat-store -fprefetch-loop-arrays -fpeel-loops -funswitch-loops -fbranch-target-load-optimize	835

It is observed that genetic algorithms determine combinations of parameters which lead to amelioration of program behaviour obtained after compilation.

6. Conclusions

Important progress in the computing field, data transmissions and multimedia permit, in the present, development of software applications with a high degree of complexity, which need increasingly more memory and processing speed. For this reason, lots of importance is given to the process of optimization.
Genetic algorithms use principles from natural genetics. These are used in solving certain difficult problems for which there are no known efficient algorithms.
In the future there would be use of parallel genetic algorithms for development of distributed applications. Each application from the network executes a normal genetic algorithm, and after a number of generations, these communicate and interchange genetic material. In the end all the nodes will transmit the best chromosome found to a central node. At the same time, for raising the performance, genetic algorithms have to be combined with other classical searching algorithms. A special role resides with the development of genetic algorithms that have to optimize the behaviour of non-homogenous architectures of software systems for which all other operating methods at source code sequence level, product structure level and resulting behaviours from compilation and link-editing have been exhausted.

References

[1] T. BÄCK - *Evolutionary Algorithms în Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Editura Oxford University Press, New York, 1996

[2] Cătălin BOJA – *Software Multicriterial Optimization*, The Proceedings of the Seventh International Conference of Informatics in Economy, May 2005, Academy of Economic Studies, Bucharest, Romania, Inforec Printing House, pp. 1068 – 1074, ISBN 973-8360-04-8.

[3] W CEDEÑO, V. VEMURI - *Database design with genetic algorithms*, Evolutionary Algorithms in Engineering Applications, Springer Verlag, 1997.

[4] L.D. DAVIS - *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991

[5] Randy HAUPT, Sue Ellen HAUPT - *Practical Genetic Algorithms*, JOHN WILEY & SONS, New Jersey, 2004

[6] H.J. HOLLAND - *Adaptation în Natural and Artificial Systems*, University of Michigan Press, 1975

[7] G. GREENWOOD, Q. ZHU - *Convergence in evolutionary programs with self-adaptation*, Evolutionary Computation, pp. 147-158, 2001.

[8] Ion IVAN, Cătălin BOJA – Empirical Software Optimization, Economic Informatics, vol. IX, nr. 2/2005, Inforec Printing House, Bucharest, 2005, ISSN 1453 – 1305, pp 43 – 50.

[9] Daniel Rivero, Julian Dorado, Juan Rabunal, Alejandro Pazos – *Using Genetic Programming for Artificial Neural Network Development and Simplification***,** Proceedings of the 5th WSEAS Int. Conf. on COMPUTATIONAL INTELLIGENCE, MAN-MACHINE SYSTEMS AND CYBERNETICS, Venice, Italy, November 20-22, 2006, ISSN 1790 – 5095, ISSN 1790 – 5117, ISBN 960-8457-56-4

[10] Ion IVAN, Cătălin BOJA – Optimization of software applications, ASE Printing House, 2007.

[11] Ion IVAN, Cătălin BOJA – *Empirical Software Optimization*, Economic Informatics Journal, vol. IX, no. 2/2005, Inforec, Bucharest, 2005, ISSN 1453 – 1305, pp 43 – 50.

[12] Mitchell Melanie – An Introduction to Genetic Algorithms, MIT Press, Massachusetts, 1999

[13] M. Holena, U. Rodemerck, T. Cukic, D. Linke, U. Dingerdissen – *Automatically Generated Problem-Tailored Genetic Algorithms for the Optimization of Chemical Processes,* Proceedings of the 6th WSEAS International Conference on Applied Computer Science, Tenerife, Canary Islands, Spain, December 16-18, 2006, ISSN 1790 – 5095, ISSN 1790 – 5117.

[14] Marius Liviu VOCHIN – Data compression software based on genetic algorithms, in Romanian, bachelor paper, Academy of Economic Studies, 2006