Synthesis of Scientific Algorithms based on Evolutionary Computation and Templates

Oleg Monakhov and Emilia Monakhova {MONAKHOV,EMILIA}@RAV.SSCC.RU
Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Pr. Lavrentieva, 6, Novosibirsk, 630090, Russia

Abstract

This work describes a new approach for the synthesis of algorithms based on given templates and a set of input-output pairs using evolutionary computation. The presented algorithm of evolutionary synthesis integrates the advantages of the genetic algorithms and genetic programming and was applied for automatically rediscovery and discovery of several computational, combinatorial and graph algorithms.

1. Introduction

In this work the problem of synthesis of an algorithm A is considered as a problem of searching for the parameters and functions of the given template T of an algorithm with the aim of optimization of a given objective function F characterized as a quality of the algorithm A. The template (skeleton (Cole, 1989; Mirenkov & Mirenkova, 1996), design pattern (Gamma et al., 1994)) is a parameterized control structure of the algorithm. The template describes a scanning order of data structures of the algorithm and defines the computational dynamics of the algorithm in space-time coordinates. The template T of the algorithm A contains parameters $P = \{p_k\}$, $k \geq 0$, which describe the values of input and local variables, parameters of the data structures, constants and some primitive operations of the algorithm. The template T also contains a set of functions (formulas) $FM = \{f_n\}$, $n \geq 0$, of the algorithm A. When giving the values of the parameters P and defining the functions FM in the template T, the algorithm $A(T, P, FM)$ is obtained. The objective function F estimates the discrepancy between the observed output data of algorithm $Y_i' = A(T, P, FM, X_i)$ and the given expected values Y_i for the given input values X_i, $1 \leq i \leq N$. The function F also should estimate the complexity of the algorithm $A(T, P, FM)$.

Thus, for the given template T and the given input-output values $\{X_i, Y_i\}$, $1 \leq i \leq N$, a problem of algorithm discovery is to find the parameters P^* and to determine the functions FM^* in the template T defining of the algorithm $A^*(T, P^*, FM^*)$ such that

$$F(A^*(T, P^*, FM^*, X_i)) \leq F(A(T, P, FM, X_i))$$

for all $1 \leq i \leq N$, $P \in Dom(P)$, $FM \in Dom(FM)$.

As a solution to the problem, a new template-based evolutionary approach is proposed for computer discovery (synthesis) of algorithms optimizing a given objective function. This approach integrates the templates, genetic algorithms (GA) (Goldberg, 1989), genetic programming (GP) (Koza, 1992) and obtains some new properties: more complex loop structure and recursion of the created algorithms than in genetic programming, and synthesis of new functions and predicates that the genetic algorithms can not create. These properties of the approach are based on the background knowledge and generalization of the expected algorithm and application field included in the template. The traditional genetic programming (GP) suffers from weak-restricted blind search in huge spaces for real-world problems and has the high time efforts. This work investigates the use of templates of algorithms to restrict the search space to admissible models and structures of solutions.This approach represents the flexible and direct way to incorporate expert knowledge about parameters, variables, functions and structures of the problems into being developed computation models. In the extreme cases, on the one hand, if we do not know a template of the algorithm, this approach gives us the traditional genetic programming. On the other hand, if we know a full structure of the algorithm and do not know only some parameters, this approach gives us the traditional genetic algorithm for searching of the optimal parameters.

A practical approach to a hybrid GA/GP search without templates and multiple trees was used in (Andre, 1994; Nguyen & Huang, 1994; Lee, Hallam & Lund, 1996). The GA performs the search to find the right

values for the constants, while the GP searches the space of parse trees. The chromosome representation contains both the parse tree and the constant values, and there are different genetic operators for manipulating the parse trees and the constants. Another interesting approach in (Olsson, 1998) to the synthesis of scientific algorithms and programs is based on evolutionary computation but without genetic operations and templates.

2. Template-based Evolutionary Algorithm for Automatic Discovery

The automatic discovery algorithm is based on evolutionary computation and the simulation of the survival of the fittest in a population of individuals, each being presented by a point in the space of solutions of the optimization problem. The individuals are presented by data structures Gen - chromosomes. Each chromosome contains underdeterminated parameters p_k and functions (formulas) f_n of the template: $Gen = \{P, FM\} = \{p_1, p_2, ..., p_k; f_1, f_2, ..., f_n\}$, $k, n \geq 0$. These parameters and functions determine the required algorithm $A(T, Gen)$ based on the given template T and the chromosome Gen.

Each population is a set of chromosomes Gen and determines a set of algorithms $A(T, Gen)$ generated based on the template T.

The main idea of the discovery algorithm consists in the evolutionary transformations over sets of the chromosomes (parameters and formulas of the template) based on a natural selection: "the strongest" survive. In our case these individuals are algorithms giving the best possible value of the objective (fitness) function. In the algorithm the starting point is the generation of the initial population. All individuals of the population are created at random, the best individuals are selected and saved. To create the next generation, new solutions are formed through genetic operations named selection, mutation, crossover and adding new elements.

The function F named as fitness function evaluates the sum of quadratic deviations of output data of algorithm $Y_i' = A(T, Gen, X_i)$ from the given expected values Y_i for the given input values X_i, $1 \leq i \leq N$:

$$F = \sum_{i=1}^{N} (A(T, Gen, X_i) - Y_i)^2 + C(A(T, Gen)),$$

where $C(A(T, Gen))$ is an estimation of the complexity of the algorithm A (a time of execution, a number of iterations, a complexity of formulas). In practice, we use a number of iterations for the estimation of the complexity in the case of synthesis of iterative algorithms, otherwise we use the sum of number of nodes (length) of formulas. The purpose of the discovery algorithm is to search for a minimum of F.

3. Data representation

Basic data structures in our program realizing the evolutionary algorithm are the chromosomes Gen.

In this work a new approach for representation of the chromosomes Gen is proposed. The chromosome Gen is based on an integration of a linear structure of the chromosome for representation of parameters p_k (as in genetic algorithms (Goldberg, 1989)) and a multi-tree structure of the chromosome for representation of functions (formulas) f_n (as in genetic programming (Koza, 1992)). The linear structure of a chromosome is used for representation of the following parameters p_k of the given template T: values of integer and real variables and constants; values of indices, increments and decrements; signs of variables, logic operations and relations, types of rounding.

The multi-tree structure of a chromosome is used for representation of the functions (formulas) f_n of the given template T. The tree corresponds to the parse tree of the function. The variables and constants of the formula f_n are represented by terminal nodes TS of the tree. The operations and primitive functions used in the formula f_n are represented by non-terminal nodes NS of the tree. Each operation (primitive function) of the formula and its operands (the arguments of the primitive function) are represented by a node and its descendant nodes in the tree. For example, in Fig. 1 the tree representation for the formula $f_n = (x + 2)/\sqrt{a * x - 5}$ has the following nodes: $TS = \{x, a, 2, 5\}$, $NS = \{+, -, *, /, \sqrt{}\}$.

Using the template T and generating the chromosomes Gen, we create an analytical expression for each function f_n and determine a value for each parameter p_k and, after that, we can already produce all evaluations and modifications of the algorithm $A(T, Gen)$. Thus, for the known values of the functions f_n and parameters p_k we can calculate the output values Y_i' of the algorithm $A(T, Gen, X_i)$ generated by evolution of the chromosomes Gen based on the given template T for the given input values X_i, $1 \leq i \leq N$. After evaluation of the algorithms A we obtain the values of the fitness function F and select the best algorithms in the population.

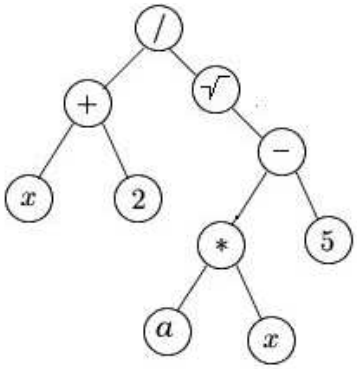

Figure 1. Tree representation of formula.

4. Operators of the Algorithm

The *mutation* operator is applied to the individuals (chromosomes) chosen randomly from the current population with a probability $p_m \in [0,1]$. Mutation represents a modification of an individual whose number is randomly selected. The modification of the linear structure of the chromosome is understood as a replacement of a randomly chosen parameter p_i by another value selected at random from a set of admissible values. The modification of the tree structure of the chromosome is performed by a replacement of the value of a randomly chosen node in the tree representation of function f_i by another value selected at random from the set of admissible values.

The *crossover* operator is applied to the two individuals (parents) chosen randomly from the current population with a probability $p_c \in [0,1]$. The crossover consists of the generation of two new individuals by exchanging the parts of the chromosomes of the parents. For the linear (or tree) structures of the chromosomes the crossover is performed by replacing a randomly chosen linear substructure (subtree) of one parent by a linear substructure (subtree) from the other parent.

The creation of a *new element* is the generation of random parameters p_k and functions f_n for the chromosomes. It allows for the adding of a diversification to the elements of a population.

The *selection operator* realizes the principle of the survival of the fittest individuals. It selects the best individuals with the minimum fitness function in the current population.

Note, only simple genetic operators were used in the algorithm, but this approach can use more complex operators developed for GP and GA.

5. Iteration Process

In the search for the optimum of the fitness function F the iteration process in the computer discovery algorithm is organized in the following way.

First iteration: a generation of the initial population. It is realized as follows. All individuals of the population are created by means of the operator *new element* (with a test and rejection of all "impractical" individuals). After filling the whole population, the best individuals are selected and saved in an array *best*.

One iteration: a step from the current population towards the next population. The basic step of the algorithm consists of creating a new generation on the basis of selection, mutation, crossover and also adding some new elements.

After evaluation of fitness function for each individual of the generation, a comparison of the value of this function to values of fitness function of those individuals which are saved in the array *best* is executed. In this case, if an element from the new generation is better than an element $best[i]$, for some i, we locate the new element on place i and shift all remaining ones per a unit of downwards. Thus, the best element is located at the top of the array *best*.

Last iteration (the termination criterion): the iterations are finished either after a given number of steps $T = t$ or after finding optimal algorithm $A(T, Gen)$ (with the given value of fitness function).

By producing a given amount of the basic steps of the template-based evolutionary algorithm, we obtain a set of algorithms $A(T, Gen)$ containing an algorithm $A^*(T, Gen)$ with the minimum fitness function F in the element $best[0]$.

6. Experimental Results

The template-based evolutionary approach was applied for rediscovery of the following algorithms: computation of the power and factorial of a natural number, finding the least (largest) element of an array, computation of the sum of the squares of elements of an array, computation of the dot product of two vectors, finding the formulas for the Fibonacci (Tribonacci) sequence, computation of the sum of matrices, bubble, merge and Shell sort, finding the roots of an equation, load balancing in parallel system, finding the single source shortest paths and minimal spanning tree in a graph. The approach was also applied for discovery of analytical descriptions of new dense families of optimal regular networks (Monakhov & Monakhova, 2003) and a distance function of circulant graphs with

degree four.

The realization of the template-based evolutionary algorithm has been implemented in the C programming language and templates have been presented in this language. The number of iterations and population size were chosen by an experimental way based on parameters from (Goldberg, 1989),(Koza, 1992).

6.1. Finding the roots of the second-order equations

For the first example, the process of rediscovery of algorithms for finding the roots of the second-order equations: $f(x) = ax^2 + bx + c = 0$ is presented. Let 20 of input-output pairs $\{X_i, Y_i\}$ be given, where $X_i = (a_i, b_i, c_i)$ are the coefficients of the equations, $Y_i = (r1_i, r2_i)$ are roots of the equations (for simplicity, we will consider only real roots), $1 \le i \le N$, $N = 20$. The following two templates are used: the first template T_1 is given as the following formula:

$$r_{1,2} = f_1(a,b,c) \pm f_2(a,b,c).$$

Note that the unknown functions are shaded.

The second template T_2 of an approximation algorithm is given as the following iterative loop:

1 $\{k = 0; x_k = x_{beg};$
2 **do** $\{x_{k+1} = f_1(x_k, f(x_k), f'(x_k))$; $k = k + 1;\}$
3 **while** $((f'(x_k) = \epsilon > 10^{-7}) \& (k < 500));$
4 **return** $x_k\}$,

with a limited number of iterations $it < 500$, with a given precision of approximation $\epsilon < 10^{-7}$ and an initial point x_{beg}, with a procedure for calculation of derivative $f'(x)$.

The terminal nodes for T_1 are $TS_1 = \{a, b, c, Cr\}$, and for T_2 are $TS_2 = \{x_k, f(x_k), f'(x_k), Cr\}$, where Cr is a set of random natural constants. The set of operations used for synthesis of the formulas is $NS = \{+, -, *, /, \sqrt{}, x^2\}$ for the both templates.

In the case of the first template the template-based evolutionary algorithm rediscovered the following known formula: $r_{1,2} = -b/2a \pm \sqrt{b^2 - 4ac}/2a$. In the case of the second template the discovery algorithm found the following expression: $x_{k+1} = x_k - f(x_k)/f'(x_k)$. This result corresponds to the known formula of Newton's method (method of tangents). The first result has been found after 10216 iterations (for time 70 sec.) with a population of 200. The second result has been found after 360 iterations (for time 10 sec.) with a population of 200.

6.2. Finding the recursive functions for the Fibonacci and Tribonacci numbers

The second example shows how the recursive function for the Fibonacci numbers can be generated from the given template and the first eight numbers with index $1 \le i \le 8$. We use the following template:

$$F(i) = f_3(F(f_1(i)), F(f_2(i))).$$

With the set of operations $NS = \{+, -, *, /\}$ and set of terminals $TS = \{i, Cr\}$ the template-based evolutionary algorithm defined $f_1(i) = i - 1$, $f_2(i) = i - 2$ and $f_3(x_1, x_2) = x_1 + x_2$ after 411 iterations (for time 12 sec.) with a population of 3000.

Tribonacci numbers can be generated from the following template:

$$F(i) = f_4(F(f_1(i)), F(f_2(i)), F(f_3(i))).$$

The template-based evolutionary algorithm defined $f_1(i) = i - 1$, $f_2(i) = i - 2$, $f_3(i) = i - 3$ and $f_4(x_1, x_2, x_3) = x_1 + x_2 + x_3$ after 461 iterations (for time 209 sec.) with a population of 30000, with the set of operations $NS = \{+, -\}$, set of terminals $TS = \{i, Cr\}$ and with the given first ten numbers with index $1 \le i \le 10$.

6.3. Finding the distance function of circulant graphs with degree four

For this example, the distance function of circulant graphs with degree 4 is created. The class of circulant networks (Bermond, Comellas & Hsu, 1995; Monakhov & Monakhova, 2000; Hwang, 2003) plays an important role in the design and implementation of interconnection networks. A circulant graph, having the parametric description, is defined as follows. *A circulant* is an undirected graph $G(N; s_1, s_2, \ldots, s_n)$ with a set of nodes $V = 0, 1, 2, \ldots, N-1$, having $i \pm s_1, i \pm s_2, \ldots, i \pm s_n \pmod N$ nodes, adjacent to each node i.

The numbers $S = (s_i)$ $(0 < s_1 < \ldots < s_n < N/2)$ are generators of the finite Abelian group of automorphisms connected to the graph. Circulant graphs $G(N; 1, s_2, \ldots, s_n)$, with the identity generator, are known as loop networks (Bermond, Comellas & Hsu, 1995). The degree of a node in circulant graph G is $2n$, where n is the dimension. We will consider loop networks with degree 4, i.e. circulant networks of the form $G(N; 1, s)$. For example, a circulant graph $C(14; 1, 6)$ with degree 4, $N = 14, s_1 = 1, s_2 = 6$ is shown in Fig. 2.

The diameter of G is defined as $d(N; S) = \max_{u,v \in V} D(u, v)$, where $D(u, v)$ (the *distance func-*

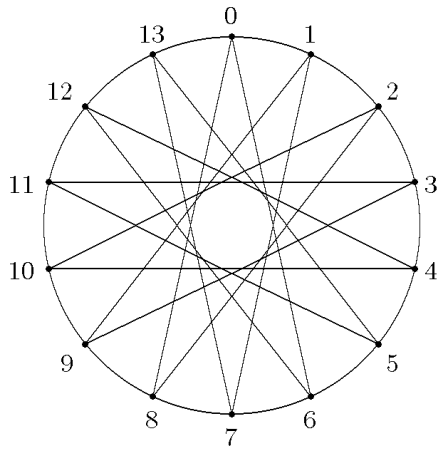

Figure 2. Circulant $C(14; 1, 6)$.

tion) is the length of a shortest path between nodes u and v in G. Because of the symmetry in circulants it is enough to consider the problem of finding a shortest path from 0 to an arbitrary node v. The distance function for nodes 0 and v in circulant $C(200; 1, s)$, $N = 200$, for $0 < s < N/2$ and $0 < v < N/2$ is shown in Fig. 3.

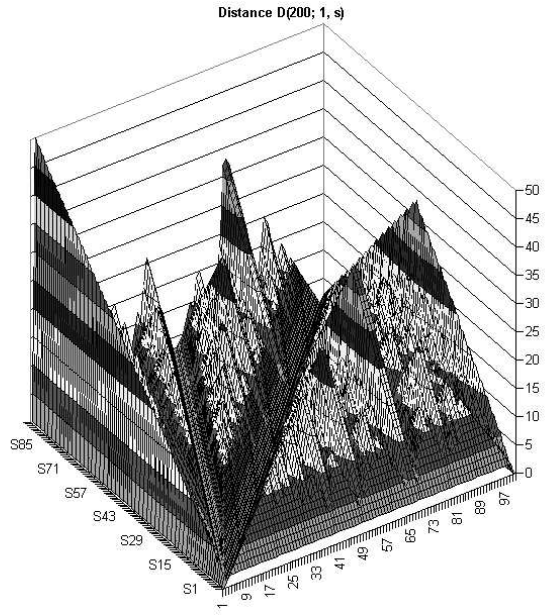

Figure 3. Distance function of circulant $C(200; 1, s)$.

For finding the distance function $D(0, v)$ we will consider the following consideration. For any node w we define $+s$ and $-s$ links from node w depending on whether they are used to go to node $(w + s) mod\ N$ (in clockwise direction) or $(w - s) mod\ N$ (in counterclockwise direction). Similarly, we define $+1$ and -1 links. Note that in circulant graphs a shortest path from 0 to v would be using at most either $(+s, +1)$ or $(+s, -1)$ or $(-s, +1)$ or $(-s, -1)$ links. Therefore further we will consider such combinations of links only. Let $(+s, +1)$-path be a path from 0 to v using $+s$ and $+1$ links only. For other combinations of links we define the analogous notations. In what follows $x/s = \lfloor x/s \rfloor$, $x\%s = x\ mod\ s$.

In order to go to node v from 0 by means of four possible ways we have to use

1) $(+s, +1)$-path: using v/s number of $+s$ links and $v\%s$ number of $+1$ links;

2) $(+s, -1)$-path: using $v/s + 1$ number of $+s$ links and $s - v\%s$ number of -1 links;

3) $(-s, -1)$-path: using $(N - v)/s$ number of $-s$ links and $(N - v)\%s$ number of -1 links;

4) $(-s, +1)$-path: using $(N - v)/s + 1$ number of $-s$ links and $s - (N - v)\%s$ number of $+1$ links.

This corresponds to one loop travelled in clockwise direction and one loop travelled in counterclockwise direction ($t = 0$). Generalizing this process for $t \geq 0$, we obtain for node v :

1) all $(+s, +1)$-paths: using $(v + tN)/s$ number of $+s$ links and $(v + tN)\%s$ number of $+1$ links;

2) all $(+s, -1)$-paths: using $(v+tN)/s+1$ number of $+s$ links and $s - (v + tN)\%s$ number of -1 links;

3) all $(-s, -1)$-paths: using $((t+1)N - v)/s$ number of $-s$ links and $((t + 1)N - v)\%s$ number of -1 links;

4) all $(-s, +1)$-paths: using $((t+1)N-v)/s+1$ number of $-s$ links and $s - ((t + 1)N - v)\%s$ number of $+1$ links.

Note that because of the symmetry of circulants the $(-s, +1)$ and $(-s, -1)$-paths from 0 to $v + tN$ can be changed to the $(+s, -1)$ and $(+s, +1)$-paths, respectively, from 0 to node $(t + 1)N - v$, $t \geq 0$.

It is necessary to find the shortest paths of all the four types and the shortest of the four will give us a global shortest path between 0 and v. The number of loops $t < s$ because $v\%s < s$ for any $0 \leq v < N$.

Based on background knowledge of circulant properties the following template T for the distance function

$dist = D(0,v)$ of circulant $C(N;1,s)$ is used:

1 **int** $t,k,k2,r,r2,d,d1,d2,dist = N;$
2 **for**$(t = 0; t < s; t = t + 1)$
3 $\{k = (v + t * N)/s; r = (v + t * N)\%s;$
4 $k2 = ((t+1) * N - v)/s; r2 = ((t+1) * N - v)\%s;$
5 $d1 =$ $f_1(k,r,s)$; $d2 =$ $f_1(k2,r2,s)$;
6 $d =$ **min**$(d1,d2)$; **if** $(dist > d)$ $dist = d;\}$
7 **return** $dist$

In the line 1 of the template the needed local variables are defined. In the line 2 we have the operator **for** with the limited number of loops $t < s$. In the lines 3 and 4 the variables k and r define the numbers of $+s$ and $+1$ links, respectively, for path from 0 to v in clockwise direction, and, similarly, the variables $k2$ and $r2$ define the numbers of $-s$ and $+1$ links for path from 0 to v in counterclockwise direction. In the line 5, for the current loop t, the undefined function $f_1(k,r,s)$ has to calculate the length $d1$ of the shortest path from 0 to v in clockwise direction, and, similarly, $f_1(k2,r2,s)$ calculates the length $d2$ of the shortest path in counterclockwise direction. In the line 6 the length d ($dist$) of the shortest path from 0 to v for the current loop t (for all loops $t' < t$) is defined. In the line 7 we have the result: $dist = D(0,v)$.

The terminal nodes for the undefined function $f_1(x_1,x_2,x_3)$ are local variables $\{k,k2,r,r2\}$, a global $\{s\}$ and $\{Cr\}$ (a set of random natural constants). The set of operations used for synthesis of the formula f_1 is $NS = \{+,-,\min,\lfloor x \rfloor\}$.

For this template the template-based evolutionary algorithm found the following expression: $f_1(x_1,x_2,x_3) = \min((x_1 + x_2),(x_1 + 1 + x_3 - x_2))$. The function $f_1(k,r,s)$ calculates the length of the shortest path from 0 to v in clockwise direction as the minimum of the lengths $(k+r)$ and $((k+1)+(s-r))$ of the $(+s,+1)$- and $(+s,-1)$-paths, respectively, and, similarly, the function $f_1(k2,r2,s)$ calculates the length of the shortest path from 0 to v in counterclockwise direction (both for the current loop t). The result has been found for the given 99 input-output pairs $\{v, D(0,v)\}$, $1 \le v \le 99$, for graph $C(200;1,s)$ after 127 iterations (for time 270 sec.) with a population of 500. The correctness for computation of the distance function $D(u,v)$ of circulant based on template T and formula f_1 was proved experimentally and theoretically.

The upper estimate of t (equal to s in the line 2 of the above template) can be decreased.

Lemma 1 *The number of loops in the algorithm defining the distance function (the line 2 of the above template) may not exceed the following value:* $\lfloor (s/2 + 1)/\lfloor N/s \rfloor \rfloor$.

As a result we have

Lemma 2 *The computation of the distance function* $D(0,v)$, $0 \le v < N$, *for loop network* $C(N;1,s)$ *based on template* T, *formula* f_1 *and with the number of loops defined by Lemma 1 is correct.*

This algorithm can be used to solve other problems in loop networks of degree 4, such as the routing problem of finding a shortest path between two nodes, or finding the diameter of a graph. For solving the first problem it is sufficient to store numbers of steps and signs of two generators giving a shortest path if the operation $dist = d$ was realized in the line 6 of the above template T.

In (Mukhopadhyaya & Sinha, 1995; Narayanan & Opatrny, 1997; Robic & Zerovnik, 2000), the algorithms of finding a shortest path between any pair of nodes in loop networks of degree 4 are given. The above algorithm generated by the template-based evolutionary algorithm differs from all known algorithms and its estimate is not worse.

7. Conclusions

The represented template-based evolutionary approach has been used successfully to automatically invent computational algorithms and for discovery of mathematical formulas for the given data sets and for the given algorithm's templates (e.g. iterations, recursions, loops and cycles), which describe the scanning of the complex data structures (matrixes, arrays, graphs, trees) and which contain the formula templates in the body. This approach can be used for synthesis of new algorithms, functions, models and solutions, which afterwards can be theoretically investigated and justified.

References

Cole, M. (1989). *Algorithmic Skeletons: Structured Management of Parallel Computation.* The MIT Press.

Mirenkov, N., & Mirenkova, T. (1996). Multimedia Skeletons and Filmification of Methods. *Proc. of The First International Conference on Visual Information Systems* (pp. 58–67). Victoria University, Melbourne, Australia.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable*

Object-Oriented Software. Addison-Wesley, Reading, MA.

Goldberg, D. E. (1989). *Genetic Algorithms, in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA.

Koza, J. (1992). *Genetic Programming.* Cambridge, The MIT Press.

Andre, D. (1994). Automatically defined features: The simultaneous evolution of 2-dimensional feature detectors and algorithm for using them. In K. Kinnear (Ed.), *Advances in Genetic Programming* (pp. 477-494). MIT Press/Bradford Books.

Nguyen, T. & Huang, T. (1994). Evolvable 3D modeling for model-based object recognition systems. In K. Kinnear (Ed.), *Advances in Genetic Programming* (pp. 459-475). MIT Press/Bradford Books.

Lee, W.P., Hallam, J. & Lund, H. H. (1996). A Hybrid GP/GA Approach for Coevolving Controllers and Robot Bodies to Achieve Fitness - Specified Tasks. *Proc. of IEEE International Conf. on Evolutionary Computation.* IEEE Press.

Olsson, J. R. (1998). Population management for automatic design of algorithms through evolution, *Proc. of IEEE International Conference on Evolutionary Computation*, IEEE Press.

Monakhov, O. & Monakhova, E. (2003). An Algorithm for Discovery of New Families of Optimal Regular Networks. *Proc. of 6th Inter. Conf. on Discovery Science (DS 2003),* Oct. 17-20, 2003, Sapporo, Japan, Lecture Notes in Artificial Intelligence, vol. 2843, (pp. 244-254). Springer- Verlag, Berlin Heidelberg.

Bermond, J.-C., Comellas, F. & Hsu, D.F. (1995). Distributed loop computer networks: a survey, *J. Parallel Distributed Comput.*, *24*, 2–10.

Monakhov, O. & Monakhova, E. (2000). *Parallel Systems with Distributed Memory: Structures and Organization of Interactions*, Novosibirsk, SB RAS Publ. (in Russian).

Hwang, F.K. (2003). A survey on multi-loop networks. *Theoretical Computer Science*, 2003, *299*, 107–121.

Mukhopadhyaya, K. & Sinha, B.P. (1995). Fault-tolerant routing in distributed loop networks. *IEEE Trans. Comput.*, *44(12)*, 1452–1456.

Narayanan, L. & Opatrny, J. (1997). Compact routing on chordal rings of degree four. In D. Krizanc and P. Widmayer, (Ed.), *Sirocco 97*, Carleton Scientific, 125–137.

Robic, B. & Zerovnik, J. (2000). Minimum 2-terminal routing in 2-jump circulant graphs. *Computers and Artificial Intelligence*, *19(1)*, 37–46.