STUDIA INFORMATICA 2003
Volume 24 Number 4 (56)

Stanisław ZAWIŚLAK, Grzegorz FREJ
University of Bielsko-Biala, Faculty of Mechanical Engineering and Computer Science

AN INFLUENCE OF PARAMETERS OF THE EVOLUTIONARY ALGORITHM APPLIED FOR THE GRAPH K-PARTITIONING PROBLEM

Summary. In the paper, the evolutionary algorithm for k-partitioning of graph is presented. Some new robust evolutionary operations are introduced replacing traditional ones. The analysis of influence of parameters on the algorithm performance is presented.

Keywords: graph k-partitioning, special evolutionary operations

WPŁYW PARAMETRÓW NA DZIAŁANIE ALGORYTMU EWOLUCYJNEGO ZASTOSOWANEGO DO PROBLEMU K-PODZIAŁU GRAFU

Streszczenie. W pracy przedstawiono zastosowanie algorytmu ewolucyjnego do problemu podziału grafu. Zastosowano nowe, skutecznie działające operacje ewolucyjne zamiast tradycyjnych. Przedstawiono analizę wpływu parametrów na działanie zaproponowanego algorytmu.

Słowa kluczowe: k-podział grafu, specjalizowane operatory ewolucyjne

1. Introduction

The problem of graph partitioning belongs to the classical graph theory problems, but it is not so widely known and considered as others, like e.g. travelling salesman, short path or coloring problems [2]. The problem of k-partitioning can be formulated as follows: (a) Let us consider the graph G(V,E), where V – set of vertices (|V| = n), E – set of edges (|E| = m); the graph is simple, finite, without multiedges and loops, (b) Divide the vertex set into k mutually disjoint subsets whose sum gives the whole set V, (c) Create the subgraphs generated by the

distinguished subsets of V, (d) Consider all possible proper divisions of V into subsets of V (i.e. neglecting an empty subset); (e) Find the partitioning for which number of edges connecting the vertices which belong to the separate subsets is minimal, taking into account the divisions mentioned in the points b, c and d.

The obtained elements of the k-partitioning are called partitions, sections, parts or components in some references. The problem is called sometimes k-cut and in the case $k = 2$ bisection.

Additional conditions, which can be added to the above formulated problem, are e.g. (c1) the minimum number of vertices in a particular subset of set V has to be greater than the lower bound or (c2) the differences of cardinalities of the subsets must not to exceed a given ε value (so called balanced partitioning). If we consider the case when n is a multiple of k, then we can consider that the obtained components are equinumerous, i.e. the special formulation of the (c2) condition for $\varepsilon = 0$. The problem can be also generalized by considering, among others, weighted graphs, called in some references networks. Taking into account all possible formulations of the graph partitioning problem, it is considered in some papers as multiobjective one.

It should be underlined that graph partitioning problem solutions have been successfully applied not only in VLSI circuit design (what is widely known) but also in the following practical tasks, like e.g. decomposition of an optimization problem [13], decomposition of a FEM grid [10] and pattern recognition procedure by means of segmentation [11].

Several algorithms to solve this NP-hard problem [3,4] are known. One of the approaches to graph partitioning problem which has been proposed is an application of evolutionary algorithms [1,4,5,6,8,9]. The earliest paper from the aforementioned list of references was published by von Laszewski who focused an attention on the special so called structural operators. The detailed study of specialized operators for this task was presented in the paper [1].This survey contains 16 operators and their descriptions. The authors considered slightly different problem of k-partitioning called by them 'generalized graph partitioning'.

The aim of this work is to study the influence of the evolutionary algorithm parameters (i.e. population size, probabilities of operators etc.) on its performance. This problem was mentioned as especially important in paper [1] but not presented due to the scope of the works assumed by the authors. In this paper, some operators proposed in references [1,5] are adopted and the detailed analysis of their importance is discussed. The analyses were made by means of own program written in the C++ language for the exemplary graph presented in the Appendix. The similar task and analyses for different graphs have been presented in the papers [8,9], where the outcomes relevant to the presented underneath were obtained but some other evolutionary operations were used.

2. Evolutionary algorithms

The term evolutionary algorithms (EA) has been introduced to distinguish them from genetic ones. The main difference is that the knowledge about the problem is represented in particular way in the second type of them. The knowledge embedded in the task is connected with encoding rules, special operators, special method of the fitness function reckoning [12] and possible repair routines. In our case (i.e. k-partitioning), the differences consist in:

- Different encoding rules i.e. digits *0* and *1* are replaced by the numbers of the set of integer numbers i.e. *1,2, ..., k* for encoding the chromosomes. Digits *0,1* are characteristic for binary representation of decimal numbers and the length of the chromosome depends on the arbitrary researcher decision, e.g. taking into account the precision of outcomes. But our code represents the problem itself i.e. *i* on the position *j* means that j-vertex belongs to i-subset of V. Therefore the length of the chromosome code depends on the number of graph vertices |V| (examples - see Chapter 4). The more detailed considerations concerning the methodology of the graph and chromosome representations have been presented in paper [12].

- Application of the special type evolutionary operators, i.e. structural mutation and crossover as well as so called operator of local optimization OLO.

- Repair procedures applied; aiming for avoidance of procedure degeneration, i.e. obtaining the chromosomes which do not represent the solution – in our case chromosomes in which some numbers from the set {1,2, ..., k} are not present. This situation can be straightforward interpreted – the partitioning consists of less than k components what can be inadmissible due to the researcher/program user arbitrary decision.

- Special fitness function reckoned not upon the explicit formula but based upon some search through the data, i.e. incidence matrix or list, depending on the graph encoding algebraic structure. It depends on the language and the additional conditions [12].

- Other activities or methods like e.g. random immigrant which application should cause avoidance of getting stuck by the algorithm in a local minimum etc.

It is worth to add that evolutionary algorithms have been recently used to some other graph theoretical problems – like graph coloring [2] and TSP (travelling salesman problem). The essence of application of EA consists in special methods of problem encoding and usage of specialized operators tailored to the particular problem. In case of TSP, the following ideas of chromosomes have been considered: adjacency, order and path representations.

It was assumed in the paper that only the additional condition (c1) is taken into account. The condition (c2) is neglected in general, but some evolutionary operations are consistent with it, e.g. structural crossover preserving some complete partitions copied from one chromosome to another. Two possible formulations of (c1) can be considered: (f1) it is forbidden to obtain the any offspring non-consistent with (c1), or (f2) it is forbidden to obtain

the whole population non-consistent with (c1). This second case is incorporated by introducing the elite to a population which fulfills this condition. Elements of this elite pass the proper solutions through the consecutive generations of the evolutionary algorithm.

3. Some specific problems – repair procedures

During execution of an evolutionary algorithm, as the result of usage of different evolutionary operators, the improper chromosomes can be obtained, i.e. representing the partition into less than k components or unbalanced partitions if such an assumption is added e.g. (c1) and/or (c2). Aiming for the situation in which such chromosomes are excluded or do not spread alongside the population in excessive quantity, a special repair algorithm should be introduced, which controls whether the chromosomes produced by particular operations are admissible.

The repair algorithm, for the proposed evolutionary algorithm, can work in the following way:

(a) After finishing of activities of all evolutionary operators applied, it is checked if all the chromosomes are acceptable, i.e. represent the partitioning into proper components.
(b) If a prohibited chromosome has been found then it is replaced by the copy of a randomly chosen member of the population.

However the following condition should be satisfied: proper performance of the repair procedure needs an ever existence of at least one proper chromosome in the population, otherwise there is a threat that the algorithm will be unable to do this.

The solutions to the above thread -proposed in references - are: (s_a) an application of special controlled initialization of the starting population, which consists only of acceptable chromosomes, (s_b) assurance that the elite size is nonzero, what guarantees survival of at least one proper chromosome from the previous base population. As was said above the second method was adopted.

4. Evolutionary operators

Based upon the above considerations and some other references of the one of the authors [8, 9, 12] it can be stated that the proper representation of the task as well as the method of introductory or initial population were chosen, i.e. (c2) is fulfilled. The next step in building the evolutionary algorithm is usually a careful design of evolutionary operators, which are relevant to the problem. In particular, the algorithms are able to acquire the knowledge

enclosed in chromosomes. It means that beside the fitness function reckoning, other control activities can be done, e.g. comparison of cardinalities of components. The so called intelligent operators were applied: structural crossover, structural mutation and the special operator of local optimization OLO [1, 8, 9]. In addition, the operator of bit type mutation (one-point mutation) was incorporated, which has been recognized as giving the effect of wider exploration.

4.1. Structural crossover

The operator of crossover is very important for the proper performance of every evolutionary algorithm. If the crossover operator destroys too much information obtained previously (i.e. in past generations) by the algorithm then it can turn into the random search algorithm.

In our case, to avoid the loss of too much information, we use the operator of structural crossover. This operator copies one full component of partitioning from the parent solution (chromosome) to the offspring. Thus, it is different from the ordinary one-point-cut crossover - which destroys the information about graph components -dividing simply a chromosome into two parts.

An analysis of results of structural crossover is presented based upon the following example. It simultaneously shows an advantage of it over one-point crossover. Let us assume that for crossover two following parents were chosen – series of 12 elements (what means that a graph consisted of 12 vertices is divided into 3 parts): $p_1 = (1\ 1\ 2\ 3\ 1\ 2\ 1\ 3\ 2\ 2\ 3\ 3)$ and $p_2 = (1\ 2\ 1\ 3\ 3\ 2\ 2\ 1\ 3\ 3\ 1\ 2)$. The divisions encoded by p_1 and p_2 are as follows: $V_1 = \{v_1,v_2,v_5,v_7\}$, $V_2 = \{v_3,v_6,v_9,v_{10}\}$, $V_3 = \{v_4,v_8,v_{11},v_{12}\}$ and $W_1 = \{v_1,v_3,v_8,v_{11}\}$, $W_2 = \{v_2,v_6,v_7,v_{12}\}$, $W_3 = \{v_4,v_5,v_9,v_{10}\}$, respectively. As the first step, a random substring from the partition is chosen (e. g. denoted by 3) and it is copied from p_1 to p_2:

$p_1 = (1\ 1\ 2\ 3\ 1\ 2\ 1\ 3\ 2\ \ 2\ 3\ 3)$

$\qquad\qquad\quad\ \downarrow \quad\ \downarrow \quad \downarrow\downarrow$

$p_2 = (1\ 2\ 1\ 3\ 3\ 2\ 2\ 1\ 3\ 3\ 1\ 2)$

as the result of the above step, the following chromosome is obtained:

$p'_2 = (1\ 2\ 1\ 3\ \underline{3}\ 2\ 2\ 3\ \underline{3}\ \underline{3}\ 3\ 3)$.

As can be seen from the above example, the copying procedure damages the demand of equal sizes of the copied part of partitioning. Third component has 7 elements in the temporary offspring because there were vertices belonging to the third component in the chromosome p_2 different than these in the first one. Repair procedures could be applied. It can be seen that in the initial p_2, there were elements representing part 3 and they have not been the elements of copied part i.e. elements 5, 9 and 10 (underlined genes). These elements are erased (the asterisks symbolize this action):

p''_2 = (1 2 1 3 * 2 2 3 * * 3 3).

In the next step, these places are randomly filled by numbers representing other parts, which were rewritten during copying procedure (underlined genes underneath):

p_2 = (1 2 1 3 3 2 2 1 3 3 1 2).

Finally, the obtained chromosome (child) – can be as follows:

p'''_2 = (1 2 1 3 1 2 2 3 2 1 3 3).

In the above described example, one chromosome was received, in the evolutionary algorithm – from two parents two children (offsprings) should be obtained, the second chromosome is created according to the same way. The division (partitioning) encoded by p'''_2 is as follows: $U_1 = \{v_1, v_3, v_5, v_{10}\}$, $U_2 = \{v_2, v_7, v_8, v_9\}$, $U_3 = \{v_4, v_8, v_{11}, v_{12}\}$. It can be seen that $V_3 = U_3$ i.e. the component 3 was preserved.

Using the standard crossover would cause loss of information about parts in a particular partitioning. Introducing the structural crossover operator has the following adventages: (i) an exchange of genes is not completely random, it preserves one partition (in some references, a jargon phrase "intelligent operator" is used), (ii) additionally it can assure equal or close numbers of vertices in every part (then additionally, if special repair procedure has to be inserted), (iii) it increases the convergence to the solution of the algorithm.

4.2. One-point mutation

The ordinary operator of bit type mutation (one-point mutation) turns one gene in a chromosome into particular random number from the set {1, 2, ..., k}. Unfortunately it changes the number of graph vertices in singled out components (first and third component in the example, underneath).

For a chromosome chosen for mutation (in the case of 3-partitioning):

p = (1 2 1 3 3 2 2 1 3 3 1 2),

one number from the interval [1, ..., n] is randomly generated, where n is the length of the chromosome (n = 12). Let it be 11, than the gene on position 11 (i.e. 1) is randomly turn into the number from the set {1,2,3} e.g. 3. It means that the third component has five elements and the first component has three elements after this operation. Like it has been said previously the elite in population is established to avoid the case when the chromosomes from the obtained population represent only the h-partitioning, where $h < k$. In our case it could happen if in next operations would cause the total removal ones from the chromosome; e.g. the same chromosome should be consecutively drawn and all ones would be turned into 2 or 3. It is probably very rare case but possible. The elite preserves, assumed in advance, number of proper chromosomes representing the k-partitioning.

4.3. Structural mutation

Additionally, other - so called - operator of structural mutation was proposed. It replaces mutually two genes (1 and 3 in the example underneath). For a chromosome chosen for mutation:

p = (1 2 1 3 3 2 2 1 3 3 1 2),

two numbers from the interval [1, ..., n] are randomly generated. In the example, these numbers are 4 and 8 which show the positions of genes for swapping (underlined genes, underneath):

p = (1 2 1 3 3 2 2 1 3 3 1 2),

in the next step, if the values chosen are different that the genes are exchanged (i.e. 1 and 3 in the example). Otherwise – the random choice of positions is repeated. Finally the child-chromosome (offspring) can be written as:

p = (1 2 1 1 3 2 2 3 3 3 1 2).

The interpretation of this operation is as follows: at the beginning we have the partitioning: $V_1 = \{v_1,v_3,v_8,v_{11}\}$, $V_2 = \{v_2,v_6,v_7,v_{12}\}$ and $V_3 = \{v_4,v_5,v_9,v_{10}\}$, after the operation: $V_1 = \{v_1,v_3,v_4,v_{11}\}$, $V_2 = \{v_2,v_6,v_7,v_{12}\}$ and $V_3 = \{v_4,v_8,v_9,v_{10}\}$. The mutual exchange of genes causes that the cardinalities of the sebsets V_i (i = 1, 2, ..., k) remains the same.

4.4. Operator of local optimization

The last evolutionary operator used in present work is an one-argument operator of local optimization (local search) OLO. This operator is essentially different in comparison to others. It acts on one chromosome but more complicated calculations are performed in comparison to all above described operators.

The OLO does not act in a blind search way. Usually evolutionary operators do modify chromosomes and they do not 'care' if the value of fitness function after their activity is lower or higher, but operator OLO, in contrary, just does it. During the execution of operations on chromosomes (see procedure described underneath), the fitness parameter is checked simultaneously. If the fitness parameter decreases then the modifications are cancelled. Therefore this operator does not turn the chromosomes into worse ones. The idea of activities performed by this operator can be compare to solutions applied in the traditional algorithms of optimization (see chapter 5.3 and 9.4 in [7]). The traditional algorithm, which takes into account consecutive points tending to the optimal one (obtained in consecutive iterations), checks the surroundings of the point looking for the steepest inclination. Then the next point is chosen in the established direction.

The operator of local optimization modifies each gene in a chromosome in the following way:

- value of the gene is randomly changed one by one into a number from the interval [1, ..., *part_num*], where *part_num* = *k* is the number of components into which the graph is divided,
- after every change, the value of the fitness function is calculated,
- if the fitness is better, then the next gene in the chromosome is analyzed, if the worsening takes place – the previous value of the gene is restored.

It should be underlined that, the speed of this operator activities depends in general on the length of chromosome *n*, number of assumed graph components *part_num* as well as the population size *pop_size*. Therefore time of execution of the operator is proportional to $n \times part\_num \times pop\_size$ and in case of greater values can substantially slow down the whole evolutionary algorithm.

All the above operators are active when the previous draw gives the result less then the probability parameter established by the user. It worth to underline that in the case of the classical genetic algorithms low values of mutation probabilities are suggested and for graph partitioning problem it is not the case.

5. Numerical analysis of influence of algorithm parameters

Values of different parameters used by the evolutionary algorithm can be set by the researcher – computer program user. Furthermore in more advanced evolutionary software environment, they can be controlled by the special supervision procedures whose aim is to adjust all the parameters properly. There are some attempts to prepare a self-adapting evolutionary software.

Because the self-adapting evolutionary programs are extremely complicated and not too easy to write, we decided to make an analysis of the effect of parameters changes in ranges set by ourselves on the algorithm effectiveness. The result will be the set of recommendations for future users which values of parameters should be taken into account. The above mentioned methodology could be an objective of further investigations (e.g. analyzing all 16 operator mentioned in [1] or their combinations). The values of chosen parameters of the algorithm are established in advance and they are not changed during the program execution. They could be adjusted experimentally, based upon the wide possible analysis of their importance e.g. changing ranges of parameters, taking into account new graphs etc. Such analyses are presented underneath in this paper and other [1,4,8,9]. The influence of the following parameters were here analyzed: population size, elite size and probabilities of

particular evolutionary operators on performance of the evolutionary algorithm by means of own computer program written in C++ language.

The best solution to the presented problem would be establishing the proper values of algorithm parameters by theoretical proof but it is impossible till now - due to lack of the tools in such young field of knowledge as the theory of evolutionary algorithms is.

In general, the theoretical proves of effectiveness of evolutionary algorithms are very rare. Only a few theorems are known e.g. about schemes. It would be a valuable achievement to do so for graph partitioning case but it exceeds the purpose of this paper. Furthermore the problem depends on the graphs considered. Other papers of one of the authors, mentioned above, where other graphs had been considered, do confirm the tendencies shown by the underneath presented figures and analyses.

5.1. Population size

In order to establish the optimal population size of the evolutionary algorithm, the following analyses were performed: investigating an influence of the population size on the running time of the program as well as on the quality of achieved outcomes. The algorithm effectiveness was measured for the following parameters: the number of generations (iterations) – 2000, probability of structural crossover – 0.3, probability of bit type mutation – 0.2, probability of structural mutation – 0.5, probability of using the operator of local optimization – 0.04, elite size – 2, the test graph g-44 (presented in Fig. 1), n = 44, number of components k = 3.

The minimum is known – it is equal 3. It is stated only for test purposes and was used for establishing the useful stopping condition for the program. The condition on balanced partitioning was not entered into the program.

The algorithm was run independently for particular population sizes in the range 10 ÷100 increasing the size by 10 (initially even by 5). The run of the computer program was finished after performing 2000 generations. The results have been presented in Fig.2. Based upon the charts, the following conclusions can be drawn: from the Fig.2a – in case of constant number of generations, the running time increases approximately in linear way with the population size in the whole assumed range, i.e.: 10 ÷ 100. However, an influence of the population size on the obtained results of the algorithm has different shape. From the chart in Fig. 2b, it can be clearly seen that for the populations enclosing less then 30 chromosomes, the algorithm after performing 2000 generations still had not found the optimal solution, the best results achieved were far from the minimum.

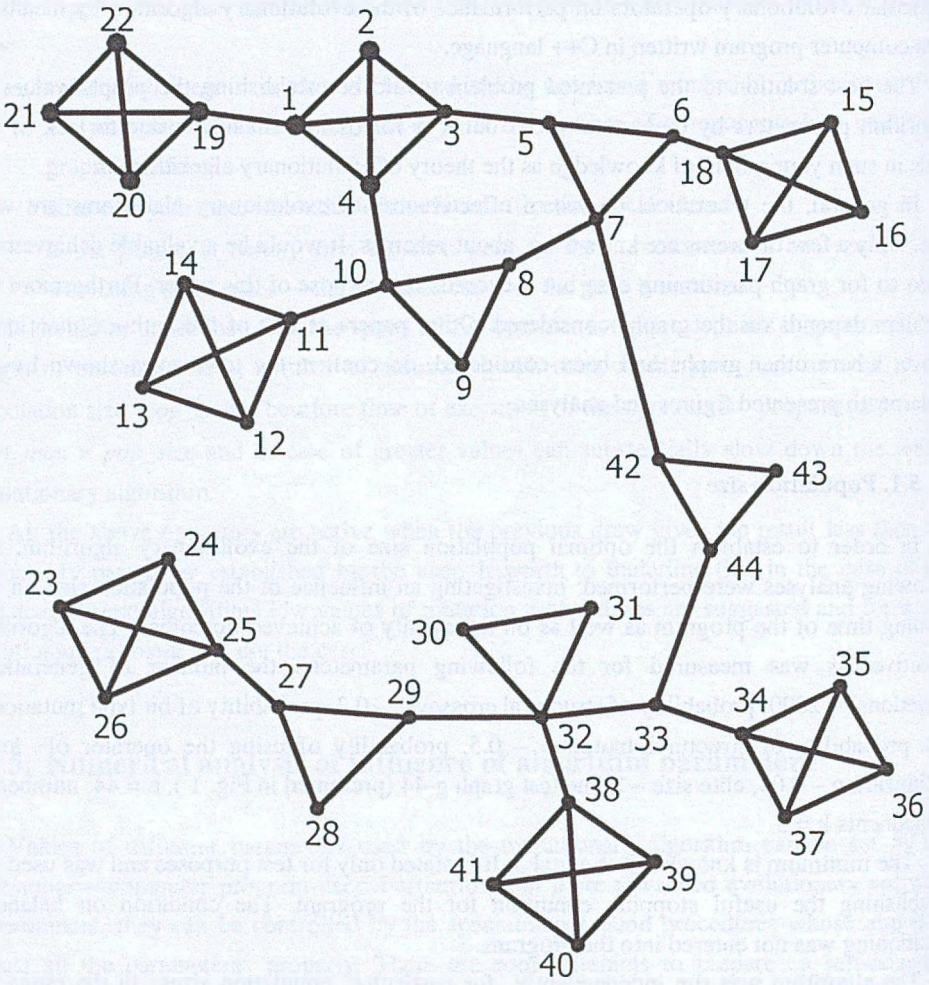
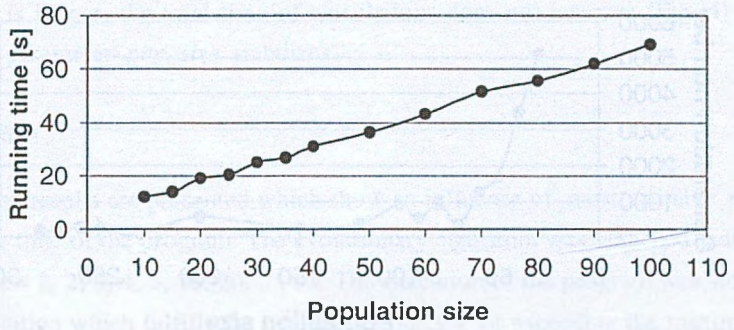


Fig. 1. Test graph g-44 which consists of 44 vertices, 70 edges, the found minimum for k-partitioning is equal to 3 (k = 3)

Rys. 1. Graf testowy g-44 o 44 wierzchołkach, 70 krawędziach; znalezione minimum dla podziału na trzy partycje wynosi 3

The program found the minimum just after increasing the population size up to 40. It should be added that further increasing of population size (*pop_size*) does not have essential effect on the improvement of obtained results, because for the population greater than 60 chromosomes – the evolutionary algorithm did not find better solutions. For example; for the data of pop_size = 80 – the running time of the program is two times longer than for pop_size = 40 (in the case in which other parameters are the same), and the reached solution is almost the same in both cases.

(a)



(b)

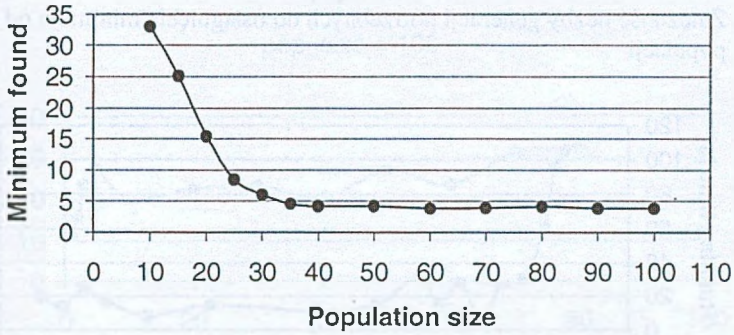


Fig. 2. Influence of population size on program running time (a) and on its effectiveness (b) for 2000 generations

Rys.2. Wpływ rozmiaru populacji algorytmu ewolucyjnego na czas obliczeń (a) i jakość osiągniętego wyniku (b) dla 2000 generacji

However it should be underlined, as it has been assumed at the beginning of the test, the evolutionary algorithm was stopped after performing of 2000 generations, even if the minimum has been found earlier. The greater the population size was the earlier the solutions were reached. Due to this, the more detailed analysis of an influence of the population size on the convergence of the evolutionary algorithm was made. The same parameters like in the previous test were used, except one of them i.e. the stopping condition. The stopping condition was established as follows: the minimum value of the cutting edges is less than or equal 4 ($\min \leq 4$) i.e. the result is close to the global minimum which for the test graph is 3. It was assumed, like in the previous one case, that the assumed in advance number of generations should be performed.

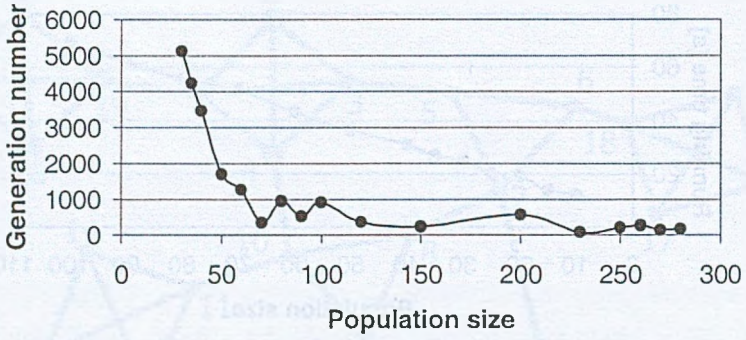


Fig. 3. Generation number as a function of population size
Rys. 3. Zależność liczby generacji potrzebnych do osiągnięcia minimum od rozmiaru populacji

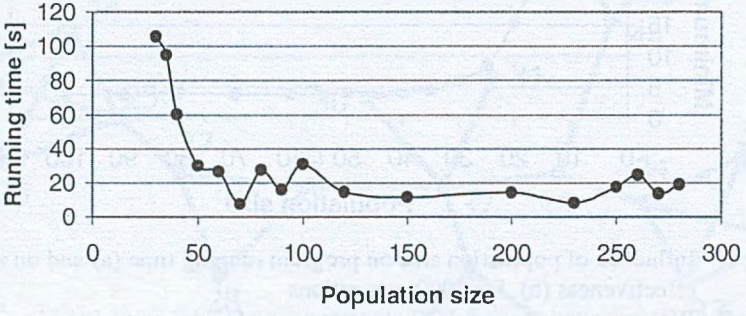


Fig. 4. Running time of evolutionary algorithm as a function of population size
Rys. 4. Zależność czasu obliczeń algorytmu ewolucyjnego od rozmiaru populacji

The population size was changed in the range 10 ÷ 280. The results are presented in Fig. 3 and 4. Analyzing Fig. 3 and 4 it can be stated that the population size has an essential influence on the convergence of the evolutionary algorithm. In the case of small populations, the algorithm results are not close to the minimum. Good results are obtained for the case when pop_size = 50, furthermore the number of generations in the algorithm needed for reaching the minimum (min = 4) is approximately 1700. The time needed for obtaining this result equals approx. 30 s. After increasing the population size up to 70 chromosomes, the convergence of the algorithm improves twice. The most interesting is that the time of reaching the acceptable results is also lower.

Comparing the charts in Fig. 2 and 4, an interesting conclusion can be drawn. Despite the fact that the greater population size is, the longer is the running time of the program but the convergence of the algorithm also increases. It means that there is purposeless to increase the population number over 60 to 70 chromosomes because it does not cause any improvement in

reached solution. Despite the fact that the time of creating and analyzing of one generation in the algorithm is longer, the total time of calculations does not increase (Fig. 4) but above some value of parameter *pop_size* stabilizes.

5.2. Elite size

In Fig. 5 the results are presented which show an influence of most suitable elite size η on the running time of the program. The evolutionary algorithm was launched independently 10 times for η = 1, 2, 3, 4, 5, 10, 20, ..., 80. The execution of the program was stopped after finding the solution which fulfills the inequality. min $\leq$ 4 or exceeding the maximal number of generations *max_gen* = 1000.

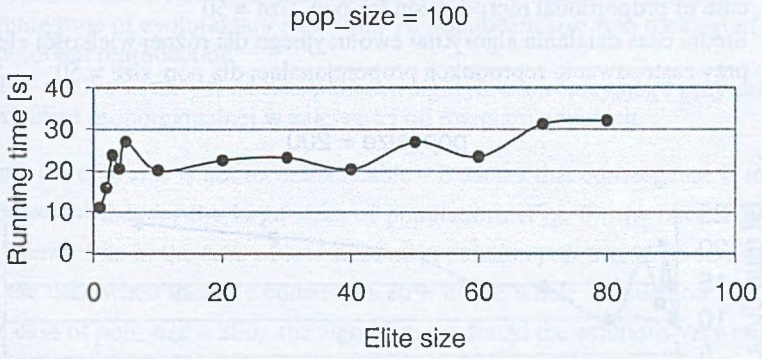


Fig. 5. Average running time of evolutionary algorithm for different elite sizes in the case of proportional reproduction for pop_size = 100

Rys. 5. Średni czas działania algorytmu ewolucyjnego dla różnej wielkości elity η przy zastosowaniu reprodukcji proporcjonalnej dla pop_size = 100

The parameters of the evolutionary algorithm used in this task: (a) probability of structural crossover – 0.3, (b) probabilities of mutation – bit type - 0.2, (c) – structural – 0.5, (d) probability of operator of local optimization – 0.04, (e) test graph g-44 (see Fig. 1), (f) number of components – 3, (g) population size *pop_size* = 100.

The tests of the algorithm were performed for pop_size = 50, max_gen = 2000 as well as pop_size = 200, max_gen = 500, for the similar probabilities of the evolutionary operators (Fig. 6 and 7).

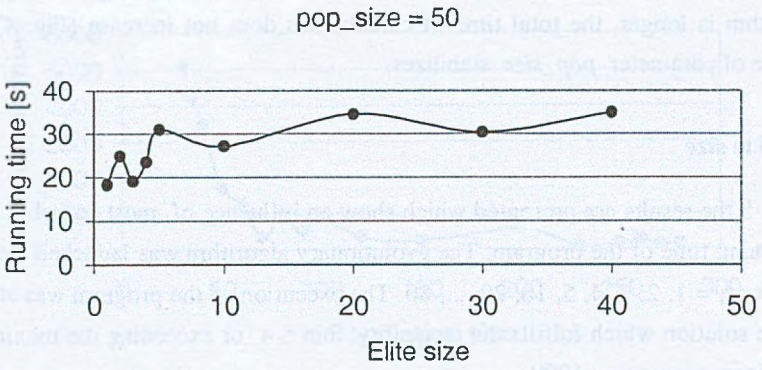


Fig. 6. Average running time of evolutionary algorithm for different elite sizes in the case of proportional reproduction for pop_size = 50

Rys. 6. Średni czas działania algorytmu ewolucyjnego dla różnej wielkości elity η przy zastosowaniu reprodukcji proporcjonalnej dla pop_size = 50

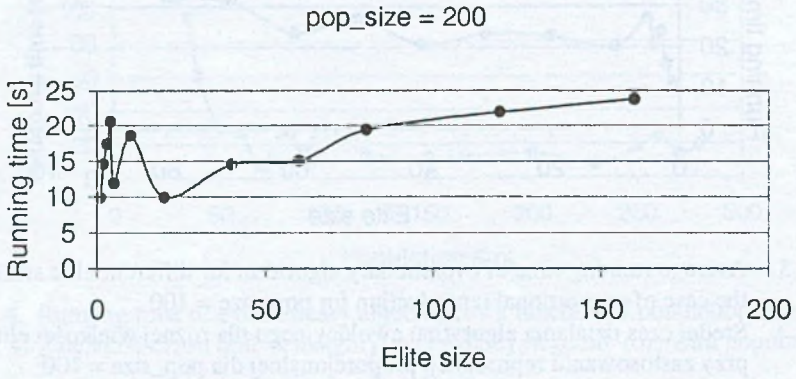


Fig. 7. Average time of evolutionary algorithm running for different elite sizes η for proportional reproduction; pop_size = 200

Rys. 7. Średni czas działania algorytmu ewolucyjnego dla różnej wielkości elity η przy zastosowaniu reprodukcji proporcjonalnej dla pop_size = 200

As can be seen in Fig. 6 and 7, for *pop_sizes* = 50 and 200, the shortest times of calculations were reached for the elite size η = 1. Similar outcomes was achieved for *pop_size* = 100, what allows us for the conclusion that the best results are obtained for the small elite (a few chromosomes).

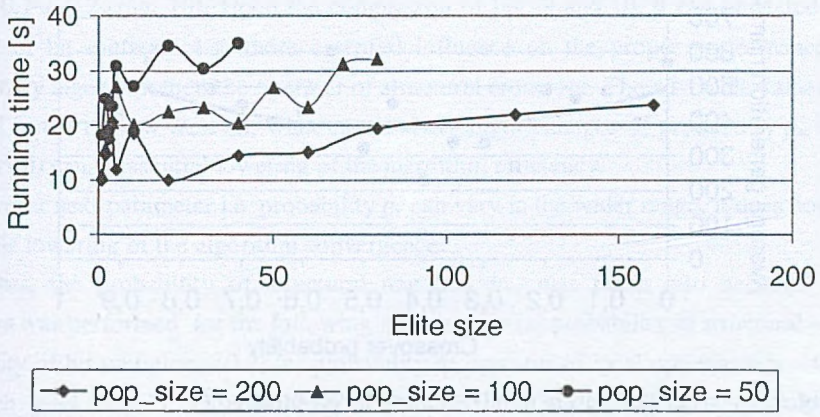


Fig. 8. Running time of evolutionary algorithm vs. population size η in the case of proportional reproduction

Rys. 8. Wpływ wielkości elity η na czas obliczeń algorytmu ewolucyjnego przy zastosowaniu reprodukcji proporcjonalnej w zależności od rozmiaru populacji

Increasing the elite size is not recommendable – it causes that convergence is too fast.

It can be stated that for the large sizes of populations (Fig. 8), the elite size has not so essential influence like in the case of low sizes of population *pop_size*. It is confirmed by the fact that in the case when the elite constitutes 80% of the whole population (i.e. very large elite), in the case of *pop_size* = 200, the algorithm has found the solutions very rarely after in advanced established number of generations. However, in the case of *pop_size* = 50 it was just impossible. The elite size should be taken considerably into account, choosing the values of the parameters.

5.3. Probabilities of evolutionary operations

Aiming for an assurance of an effective execution of the evolutionary algorithm, the probabilities of incorporating of particular evolutionary operators should be properly chosen – in our case - special types of crossover and mutation operators.

Analyzing an influence of the structural crossover probability p_c on execution of the algorithm the following tests were done. The algorithm run independently 10 times for every from the following values of the probability p_c: 0, 0.05, 0.1, ..., 0.9. The values of other parameters were listed below: (a) probability of bit type mutation – 0.2, (b) probability of structural mutation – 0.5, (c) probability of operator of local optimization – 0.04, (d) test graph g-44 (Fig. 1), (e) number of components k = 3, (f) population size *pop_size* = 100, (f) performance of the algorithm has been stopped after finding the value min ≤ 4 or after exceeding the number of generation higher than 1000.

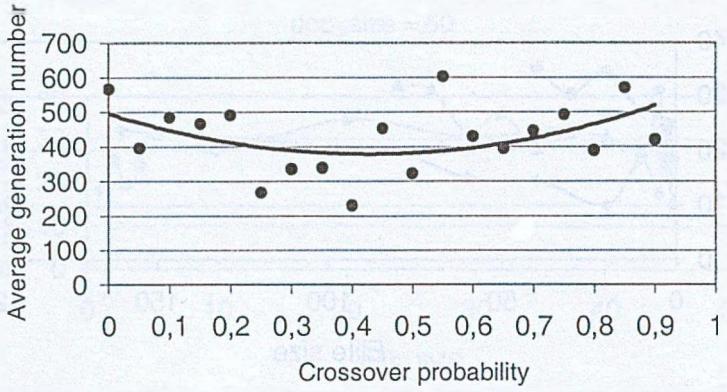


Fig. 9. Average generation number vs. crossover probability
Rys. 9. Średnia liczba generacji jako funkcja prawdopodobieństwa krzyżowania strukturalnego

From the chart in Fig. 9, it can be seen that the best outcomes are obtained for the crossover probability p_c in the range 0.3 ÷0.5.

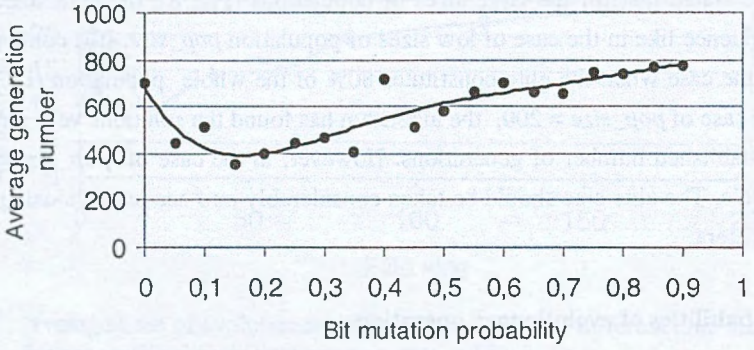


Fig. 10. Average generation number in the evolutionary algorithm vs. probability of bit (one-point) mutation
Rys. 10. Średnia liczba pokoleń jako funkcja prawdopodobieństwa mutacji bitowej (jednopunktowej)

In the next step, an influence of the probability of bit type mutation p_m on algorithm execution was analyzed Fig. 10. The tests were done for the parameters: (a) probability of structural crossover – 0.4, (b) probability of structural mutation – 0.5, (c) probability of operat of local optimization – 0.04, (d) test graph g-44 (Fig. 1), (e) partition into k = 3 components, (f) population size *pop_size* = 100, (g) algorithm performance was stopped after finding the value min ≤ 4, or after exceeding the number of generations over 1000.The course of the average generation number vs. probability of bit mutation p_m has a different

type than in the case of p_c. For the test graph – the advisable value p_m is enclosed in the interval 0.1 ÷ 0.2 (Fig. 10). Upon the comparison of Fig. 9 and 10, it can be stated that the operator of bit mutation has more essential influence on the proper performance of the evolutionary algorithm then the operator of structural crossover. The advisable values p_m are enclosed in the narrow interval, what causes that slight changes of probability p_m (leaving this interval) cause essential lowering of the algorithm efficiency.

However next parameter i.e. probability p_c can vary in the wider range, it does not cause a noticeable lowering of the algorithm convergence.

Further, the probability of structural mutation p_{mis} was taken into account. Test of algorithm was performed for the following parameters: (a) probability of structural – 0.4, (b) probability of bit mutation – 0.2, (c) probability of operator of local optimization – 0.04, (d) test graph g-44 (Fig.1), (e) partitioning of graph into k = 3 components, (f) population size *pop_size* = 100, (f) performance of algorithm was stopped after finding value min ≤ 4 or exceeding number of generations over 1000 generations.

It is worth to add, that in traditional genetic algorithms, the suggested range of mutation is lower than 0.1 or even 0.05 but these numerical investigations proved that standard suggestions sometimes do not be adequate to evolutionary operations and graph optimization problems.

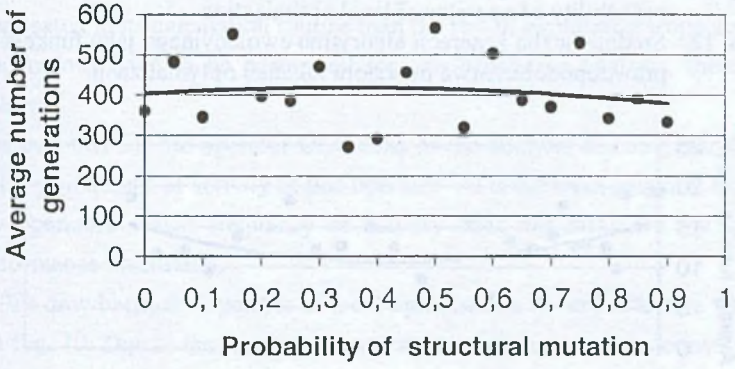


Fig. 11. Influence of probability of structural mutation on on the evolutionary algorithm running

Rys. 11. Średnia liczba pokoleń w algorytmie ewolucyjnym jako funkcja prawdopodobieństwa mutacji strukturalnej

From Fig. 11 follows that changes of parameter p_{mis} in the range 0 ÷ 0.9 were done, but any clear effect can not be recognized. It can be stated that the operator of structural mutation almost has not any influence on the performance of the proposed evolutionary algorithm or its influence is nonessential therefore it can be omitted. This conclusion corresponds to the

considered test graph. Nevertheless, in the next tests last decision was not introduced into practice.

The last performed test was an analysis of influence of the probability of the operator of local optimization p_{olo} on the effectiveness of the proposed approach, Fig.12. The algorithm was tested for the same parameters as previously: (a) probability of bit mutation – 0.2, (b) test graph g-44, (c) partitioning of graph into k = 3 components, (d) population size *pop_size* = 100, (e) performance of algorithm was stopped after finding value min ≤ 4 or exceeding number of generations over 1000 generations.

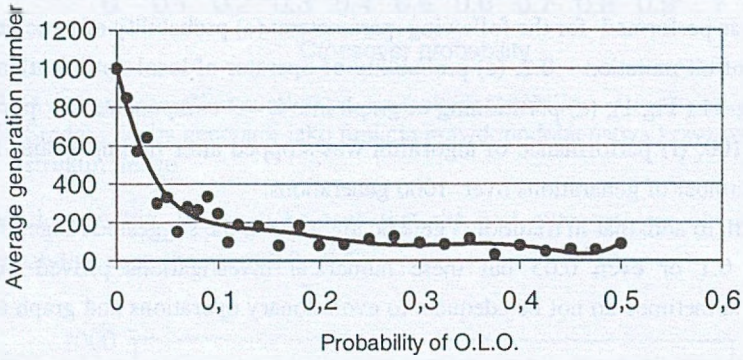


Fig. 12. Average number of generation in the evolutionary algorithm vs. probability of operator of local optimization
Rys. 12. Średnia liczba generacji algorytmu ewolucyjnego jako funkcja prawdopodobieństwa operatora lokalnej optymalizacji.

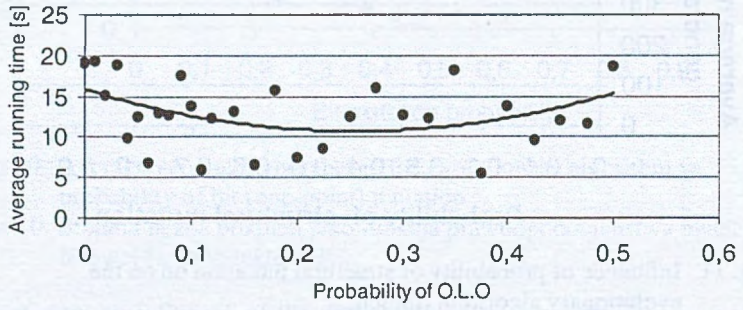


Fig. 13. Average time of calculations vs probability of operator of local optimization
Rys. 13. Średni czas obliczeń jako funkcja prawdopodobieństwa operatora optymalizacji lokalnej

Analysing the results in Fig. 12, it is easy to spot that the operator of local optimisation gives very good results. It can be seen that without the operator OLO (probability equal to 0), the tested evolutionary algorithm does not find a solution – it just stops after performing of 1000 generations. However in the case of an application of this operator with the probability

equal to 0.02, the algorithm finds fairly acceptable solutions. By increasing the probability p_{olo} up to value 0.1, it can be observed five times reduction of number of the generations repeated in the algorithm run in comparison to the variant without the OLO operator.

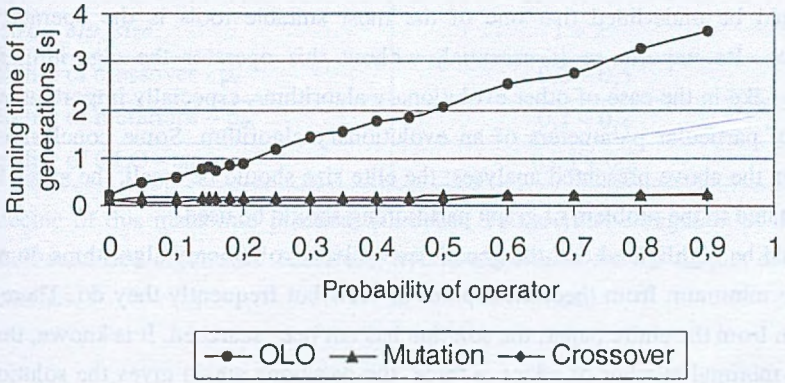


Fig. 14. Running time of 10 generations vs. the probabilities of evolutionary operators
Rys. 14. Zależność czasu obliczeń 10 generacji algorytmu od prawdopodobieństw operatorów genetycznych (ewolucyjnych)

Fig. 13 shows that it is not any essential influence of the OLO probability on the average running time of the program.

It should be fairly add that there are some costs of using this operator. The operator OLO is more complicated in its calculation routine then the rest of evolutionary operators, it means that more operations have to be performed for one generation analysis, therefore it runs essentially slower.

Fig. 14 shows that for the operator OLO time of the analysis of one generation depends linearly on the probability of activity of this operator – it is different situation from all others evolutionary operators which frequency of activity does not influence the speed of the program performance essentially.

Despite this drawback, the operator of local optimisation is very effective what is shown especially in Fig. 12. Due to the fact that the operator OLO improves the convergence of the evolutionary algorithm, the longer performance is acceptable. Fig. 13 shows that the operator of local optimisation improves the average time of the calculation approx. 30%; for the most suitable values of its probability in the range $0.1 \div 0.2$.

6. Conclusions and remarks

In the paper, the analysis of the parameters of the evolutionary algorithm applied for the graph partitioning problem has been presented. The algorithm proposed uses the special

evolutionary operators constructed especially for the task of graph partitioning. The problem is discrete and the fitness function is calculated upon the special search through the applied graph representing algebraic structure (e.g. incidence matrix, list, tree). These methods for k= 2 were described, among others, in paper [12].

It should be underlined that one of the most suitable tools is the operator of local optimisation. Its importance is essential, without this operator the algorithm is almost powerless. Like in the case of other evolutionary algorithms, especially important is a proper selection of particular parameters of an evolutionary algorithm. Some conclusions can be drawn upon the above presented analyses: the elite size should be small, the special operator OLO dedicated to the problem of graph partitioning should be used.

It should be highlighted that the genetic as well as evolutionary algorithms do not assure finding the minimum from theoretical point of view but frequently they do. Therefore, like can be seen from the entire paper, the solution has not been searched. It is known, that for test graph, the minimal number of edges is three, the partitions which gives the solution can be distinguished by the reader. The aim of our consideration was to investigate the influence of parameters, therefore even in some stopping conditions of the algorithm it has not been assume reaching the minimal number of edges connecting the parts of the particular partitioning.

Some recommended values of the parameters are listed in Table 1. Similar results were obtained for different graphs but they are not included in this paper (see some other works[1,8,9]). In overall, the result does not depend on the graph but the number of vertices should not exceed hundreds.

Further investigations for very large graphs are in progress: prospective topics could be e.g. hybrid algorithms in which more knowledge from graph theory would be embedded in evolutionary operations or parallel algorithms.

There are some suggestion that the ranks of the graph vertices can be taken into account constructing the operators. The vertices of the maximal ranks should be distributed throughout the components or collected in one component. It should depend on the distance between them in a graph.

Table 1

Recommended values of parameters of the evolutionary algorithm

Parameter	Recommended range
Population size – *pop_size*	60 ÷ 100
Elite size – *elit_size*	1 ÷ 2
Probability of crossover - p_c	0,3 ÷ 0,5
Probability of mutations – p_m	0,1 ÷ 0,2
Probability of OLO - p_c	0,05 ÷ 0.2

Introducing of this ideas into practice will cause the essential elongation of the running time of such enhanced algorithm. Therefore, hybrid algorithms are the subject of interest of some researches.

REFERENCES

1. Chmiel W., Kadłuczka P.: Special genetic operators for the problem of graph partitioning (In Polish), Algorytmy Ewolucyjne i Optymalizacja Globalna, Politechnika Warszawska, pp. 37-44, Warszawa 2000.
2. Discrete Optimisation. Models and methods of graphs colouring (In Polish). Editor M. Kubale, WNT, Warszawa 2002.
3. Falkner J., Rendl F., Wolkowicz H.: A computational study of graph partitioning, Mathematical Programming, Vol. 66, pp. 211-239, 1994.
4. Kadłuczka P., Wala K.: Tabu search and genetic algorithms for the generalised graph partitioning problem. Control and Cybernetics, V.24, No 4, pp. 459–476, 1995.
5. Laszewski von G.: Intelligent structural operators for the k-way graph partitioning problem, Proceedings of 4th Inter. Congr. On Genetic Algorithms, 1991.
6. Lin-Ming J., Shu-Park Ch.: A genetic approach for network partitioning . Int. J. Computer Math., Vol. 42, pp. 47 – 60, 1992.
7. Seidler J., Badach A., Molisz W.: Methods of optimization (in Polish). WNT, Warszawa 1980.
8. Wojnarowski J., Zawiślak S. : Evolutionary Algorithm Applied for Graph Partitioning (in Polish), in the book 'Polioptymazacja i Komputerowe Wspomaganie Projektowania', (ed. W. Tarnowski, T. Kiczkowiak) WNT, pp. 277 – 284, Warszawa 2002.
9. Wojnarowski J., Zawiślak S., Kozik S: Application of evolutionary algorithm for graph. k-partitioning (in Polish) ZN Wydziału Mechanicznego Nr 32. Polioptymalizacja i CAD, Mielno 2003, Politechnika Koszalińska, p.143-150, 2003.

10. Yang D-L., Chung Y-Ch, Chen Ch-Ch., Liao Ch-J.: A Dynamic Diffusion Optimization Method for Irregular Finite Element Graph Partitioning , The Journal of Supercomputing, 17, 91-110, 2000.
11. Yu Stella X., Gross R., Shi J.: Concurrent Object Recognition and Segmentation by Graph Partitioning , private communicate.
12. Zawiślak S., Wojnarowski J., Jagosz A.: Comparison of graph representation methods for graph partitioning problem. Implementations in some algorithmic languages. Zeszyty Naukowe ATH w Bielsku-Białej, Nr 4, Zeszyt 3, Bielsko-Biała 2002.
13. Zawiślak S., Ziemska I.: Graph theoretical approach to decomposition problem in optimal design (in Polish), Politechnika Łódzka, Bielsko-Biała, pp.159-161, 1999.

Recenzent: Prof. dr hab. inż. Zbigniew Czech

Wpłynęło do Redakcji 1 maja 2003 r.

Omówienie

W pracy omówiono problem k-podziału grafu, który ma liczne zastosowania praktyczne. Problem sformułowano warunkami (a) – (e) w rozdziale 1. Algorytmy rozwiązania tego problemu były stosowane do: projektowania układów scalonych o dużej integracji, podziału siatki modelującej obiekty w metodzie MES, rozpoznawania obrazów oraz dekompozycji dużych zadań optymalizacji konstrukcji. Dla dużych grafów (pojęcie umowne) w literaturze analizuje się różne algorytmy k-podziału grafu, przy czym w ostatnich latach wielokrotnie proponowano zastosowanie algorytmów ewolucyjnych do rozwiązania tego zagadnienia. W wielu publikacjach nie analizowano wpływu parametrów algorytmu ewolucyjnego na efektywność metody. W niniejszym artykule zaproponowano algorytm ewolucyjny do k-podziału grafu wykorzystując wybrane operacje ewolucyjne z wielu przeanalizowanych prac. Wybrano specjalizowane operatory, między innymi: mutację strukturalną (zachowuje liczności komponentów), krzyżowanie strukturalne (zachowuje jeden komponent) oraz operator Optymalizacji Lokalnej [O.L.O.](działa na pewnym etapie systematycznie, co odróżnia go od wszelkich innych operatorów). Do testów wybrano graf przedstawiony na rys. 1 o 44 wierzchołkach, przy czym autorzy przeprowadzali testy dla innych grafów uzyskując analogiczne rezultaty. Przedstawiono analizę wpływu następujących parametrów algorytmu ewolucyjnego na jego działanie: rozmiar populacji (rys. 2 ÷ 4), wielkości elity (rys. 5 ÷ 8), prawdopodobieństwa krzyżowania (rys. 9), prawdopodobieństwa mutacji (rys. 10 ÷ 11), prawdopodobieństwa operatora O.L.O. (rys. 12 ÷ 13) oraz porównano wpływ prawdopo-

dobieństw kilku operacji na czas pracy algorytmu (rys. 14). Ponieważ chodziło o testowanie działania algorytmu, więc stosowano także jako kryterium zatrzymania: zbliżanie się bieżącego rozwiązania do znanego minimum. Na podstawie analiz zestawiono zalecane zakresy parametrów algorytmu (tab. 1). Należy zauważyć, że operacja mutacji odgrywa w tym algorytmie ważną rolę i jej zakres powinien być zmieniony w stosunku do zwykle zalecanych w literaturze zakresów ($p_m < 0,05$). Zastosowanie operatora O.L.O. istotnie przyspiesza dochodzenie do rozwiązania przez przedstawiony algorytm ewolucyjny.

Adresy

Stanisław ZAWIŚLAK: Akademia Techniczno-Humanistyczna, Wydział Budowy Maszyn i Informatyki, ul. Willowa 2, 43-309 Bielsko-Biała, Polska, szawislak@ath.bielsko.pl.
Grzegorz FREJ: Akademia Techniczno-Humanistyczna, Wydział Budowy Maszyn i Informatyki, ul. Willowa 2, 43-309 Bielsko-Biała, Polska.