I Krajowa Konferencja
Algorytmy Ewolucyjne
Murzasichle, 12-15.06.1996

TWO STAGE GENETIC SEARCH ALGORITHM FOR GENERALIZED GRAPH PARTITIONING PROBLEM [1]

Piotr Kadłuczka*, Konrad Wala**
Institute of Automatics, University of Mining and Metallurgy,
Kraków, al. Mickiewicza 30, Poland
*pkad@ia.agh.edu.pl, **kwa@ia.agh.edu.pl

Abstract

The paper considers the generalized problem of partitioning the n nodes of a weighted graph into m disjoint subsets of bounded size, such that the objective function related to the weights of the graph edges is maximized. There are reported the computer experiment result of examination of approximate algorithm based on general purpose search strategy called genetic algorithm.

1. Introduction

Graph partitioning serves as a model for several important problems. For instance, it can be used in integrated circuit layout in Very Large Scale Integration [1], in storing or processing large computer programs in distributed computing systems [4],[6] and in group technology classification problems [3],[5].

The generalized graph partitioning problem (GGP), first described in [3],[4], can be formalized as the following combinatorial optimization problem. Given an undirected and weighted graph G=(N,E) and a set of numbers $\{b_1, b_2, \ldots, b_m\}$, where $b_i > 0$ for i=1,...,m. Find a graph partition $X = (X_1, X_2, \ldots, X_i, \ldots, X_m)$, $X_i \subset N$, of the set N={1,2,...,j,...,n} which maximizes:

$$f(X) = \sum_{i=1}^{m} \sum_{j,k \in X_i} c_{jk} \qquad (1)$$

subject to:

$$\bigcup_{i=1}^{m} X_i = N, \quad X_i \cap X_k = \varnothing \quad \text{for} \quad i \neq k \qquad (2)$$

$$\sum_{j \in X_i} a_{ij} \leq b_i, \qquad \text{i=1,...,m} \qquad (3)$$

where:

$a_j = (a_{1j}, a_{2j}, \ldots, a_{ij}, \ldots, a_{mj})$: weight vector of the node j∈N,

c_{jk}: weight of the edge jk∈E.

Constraints (2) ensure that each graph node is allocated precisely in one cluster (set) X_i. Constraints (3) limit the size for each cluster. The objective function $f(X)$ is the aggregate measure accounting for intra-cluster similarities or proximities.

It is assumed, without loss of generality, that for each jk∈E: $c_{jk}>0$ and for jk∉E: $c_{jk}=0$ as well as $a_{ij} = \infty$ if node j can not be assigned to cluster X_i, otherwise: $0 < a_{ij} < \infty$ for i=1,...,m; j=1,...,n. Let F(j) denote the set of feasible cluster numbers for node j : F(j)={ i∈{1,2,...,m}:

1 The work was supported by Polish Committee for Research, grant 3 P403 012 06

$a_{ij}<\infty\}$. In this formulation, b_i is the resource or size of i-cluster. If for each i and j $a_{ij}=1$ then b_i limits the number of nodes in cluster i and the constraint (4) has the form $|X_i| \leq b_i$.

We called the formulation (1),(2) and (3) the generalized graph partitioning (GGP) as opposed to standard graph partitioning where for each i: $a_{ij}=a_j$ and the constraint (3) has the form $\sum_{j\in X_i} a_j \leq b_i$. Feo and Khellaf [2] proved that graph partitioning is NP-hard combinatorial problem.

Let us define s_i, the capacity slack of i-clusters as follows. Let X be a given solution of the GGP that satisfies the assignment constraints (2) but does not necessarily meet the capacity restrictions (3), then

$$s_i = b_i - \sum_{j\in X_i} a_{ij} \quad i=1,...,m \tag{4}$$

Our genetic algorithm assures that the assignment constraints (2) are always satisfied; however, infeasibility may occur due to violation of the capacity constraints (3). Therefore, infeasible solutions are these for which s_i is strictly less than zero for at least one i. A measure of infeasibility, v, of an assignment-feasible solution X, may be defined as the absolute value of the sum of all the negative capacity slacks:

$$v(X) = \sum_{i=1}^{m} \min\{s_i, 0\} \tag{5}$$

where, to be precise, the solution $X=(X_1,X_2,...,X_i,...,X_m)$ is an assignment-feasible one if the constraints (2) are satisfied and for each $j \in X_i$: i∈F(j).

Taking into account genetic algorithm requirements concerning the solution form of the investigated problem, we introduce the second form of the GGP problem solution $x=(x_1,x_2,...,x_j,...,x_n)$, where component $x_j, x_j \in F(j)$, defines the cluster number for the node j and $X_i = \{j: x_j = i\}$. Further, we denote the GGP problem solution by symbol x corresponding to X.

2. BREADTH_DEPTH genetic search process

Genetic algorithm (GA) consists of a population and of the genetic operators. The knowledge, accumulated in the search process for the solution, is encoded as a set P of solutions called population, where M=|P| is the size of the population. The evolution process of the population P, modeled by GA, realizes the search process by use of genetic operators. Every genetic operator generates new solutions called offspring on the basis of old solutions called parents.

In the classical GA the search process is organized as the following evolution of the population. At each iteration t of the algorithm the whole 'old' population P=P(t-1) is replaced by the 'new' one P=P(t). A new population P(t) is formed by selecting the more fit solutions from the old population P(t-1) and some (only some!) members of the new one undergo transformations by means of genetic operators to form new solutions.

Michalewicz in his works (see, Chapter 4 in [7]) exploits the modified approach to GA, modGA. The modification with respect to the classical genetic algorithm is that in the modGA a new population is formed by selecting, from the old population, independently only r, 1<r<M, solutions to be parents and r solutions to die. These selections are performed with respect to the fitness of the solutions: a solution with a better than average performance has a higher chance to be selected as parent, solutions with a worse than average performance have higher chances to be selected to die. Then the new population consists of (M-r) solutions of the

old population (all solutions exept these selected to die) and r offspring of the r selected parents; i.e. r (all!) selected solutions undergo transformations by means of genetic operators. This way, in course of one iteration (one generation) only r, $r < M$, solutions are selected and processed.

Michalewicz emphasizes few differences between classical GA and modGA. Firstly, modGA better utilizes the available storage resource: population size. The algorithm avoids leaving exact multiple copies of the same solution in the new populations. Additional feature of modGA is better time complexity of the whole algorithm in comparison with the classical one. Secondly, it applies genetic operators on the whole solutions as opposed to individual components of the solution. This would provide an uniform treatment of all operators used in the algorithm. Thirdly, only few, $r < M$ members of the population are changed within each generation, so the modGA belongs to a class of srteady state GAs..

We examine genetic search process called BREADTH_DEPTH (see, [8]), where in course of one iteration initially one genetic operator is randomly chosen and then r, $r \in \{1,2\}$, solutions are selected from the population and processed: one solution if unary operator is chosen and two in case of crossover operator. Thus the algorithm also belongs to the class of Steady State GAs and in this way it has all features, above-mentioned, of the modGA.

We assume that the population set P is linearly ordered in accordance with the solution's fitness by means of ordering rule BEST-WORST: the best population solution (x_{best}) has number 1,..., the worst one (x_{worst}) has number M=|P|. Besides, to simplify genetic search process and save the computation time, the parents are selected from the population P by means of random-uniform sampling mechanism and the generated offspring is placed in the population P only in case if it is better, in accordance with the rule BEST-WORST, than the worst population solution x_{worst}; otherwise the offspring is rejected.

Let us notice that the best population solutions have the chance to be selected more times than other solutions and in this way these solutions become super individuals. Such super individual has a large number of offspring and due to the constant size of the population prevents other individuals from contributing offspring in the next generations. In some generations a super individual can eliminate desirable 'chromosomal material' and cause rapid convergence to the local optimum. To avoid this shortcoming we introduce the tabu mechanism. We assume that at the first stage of the genetic search process called BREADTH some number of best population solutions, TOPTAB top-solutions, are tabu and can not be chosen as parents. It is most likely that in this case the genetic search process can be performed in the whole search space. In the second stage of genetic search process called DEPTH , after when βK offspring are generated, where $\beta<1$ and K is the number of generated offspring in one computer experiment, the genetic search process is enhanced to the space regions connected with the best population solutions for local tuning. Thus, in this stage, the mutation operator is replaced by local opimization (hill climbing) procedure and some number of worst population solutions, ENDTAB end-solutions, are tabu can not be chosen as parents.

3. Algorithm

The GENGGP3 algorithm realizes the BREADTH_DEPTH genetic search process for generalized graph partitiong problem. Algorithm exploits five genetic operators:

1. Random mutation operator RM

Operator RM generates one offspring x^1 on the basis of one parent $x = (x_1, x_2, \ldots, x_{j-1}, x_j, \ldots, x_n)$:

a) choose one number j , $1 \le j \le n$ and x_j^1=RANDOM_UNIFORM(F(j))

b) set $x^1 = (x_1, x_2, \ldots, x_{j-1}, x_j^1, \ldots, x_n)$

2. Search operator S1

Operator S1 generates one offspring x^1 on the basis of one parent $x = (x_1, x_2, \ldots, x_{i-1}, x_i, \ldots, x_{j-1}, x_j, \ldots, x_n)$:

a) choose one number i , $1 \le i \le n$, and "best" number j:

$j = \arg \max_{k \neq i} \{f(x_1, x_2, \ldots, x_{i-1}, x_k, x_{i+1}, \ldots, x_{k-1}, x_i, x_{k+1}, \ldots, x_n): k \in \{1,2,\ldots,n\}$ and v(x) is minimal$\}$

b) swap element x_i of the parent for element x_j: $x^1 = (x_1, x_2, \ldots, x_{i-1}, x_j, \ldots, x_{j-1}, x_i, \ldots, x_n)$

3. Search operator S2

Operator S2 generates one offspring x^1 on the basis of one parent $x = (x_1, x_2, \ldots, x_{j-1}, x_j, \ldots, x_n)$

a) choose number j , $1 \le j \le n$ and the "best" number i, such that

$f(x_1, x_2, \ldots, x_{j-1}, i, x_{j+1}, \ldots, x_n) = \max_{k \neq x_j} \{f(x_1, x_2, \ldots, x_{j-1}, k, x_{j+1}, \ldots, x_n): k \in F(j)$ and v(x) is minimal$\}$

b) replace element x_j of the parent with number i: $x^1 = (x_1, x_2, \ldots, x_{j-1}, i, \ldots, x_n)$

4. Random crossover operator RX

Operator RX generates two offspring x^1, x^2 on the basis of two parents $x' = (x_1', x_2', \ldots, x_j', x_{j+1}', \ldots, x_n')$, $x'' = (x_1'', x_2'', \ldots, x_j'', x_{j+1}'', \ldots, x_n'')$

a) choose one number j, $1 \le j < n$

b) generate two offspring:

$x^1 = (x_1', x_2', \ldots, x_j', x_{j+1}'', \ldots, x_n'')$, $x^2 = (x_1'', x_2'', \ldots, x_j'', x_{j+1}', \ldots, x_n')$

5. Local optimization operator LO

Let $S(x) = \{x': x' = (x_1, x_2, \ldots, x_{j-1}, x_j', x_{j+1}, \ldots, x_n): x_j' \neq x_j,\ x_j' \in F(j),\ j \in \{1, \ldots, n\}\}$ denotes the neighbourhood of the solution $x = (x_1, x_2, \ldots, x_j, \ldots, x_n)$.

Operator LO generates one offspring x^1 on the basis of one parent x:

a) find a solution $x' = \arg \max\{f(\hat{x}): \hat{x} \in S(x), \hat{x} \neq x$ and v(x) is minimal$\}$

b) if $f(x') > f(x)$ then set x:=x' and go to step a), otherwise return x^1:=x.

Each genetic operator has the operation "choose" realized by RANDOM_UNIFORM procedure. Besides, operators S1 and S2 have choose operation together with simple search operation: "choose the best number".

Algorithm: GENGGP3

To determine the approximate solution x_{approx} do the following.

STEP 1.

Generate M solutions $x = (x_1, x_2, \ldots, x_j, \ldots, x_n)$, where for each j:

$x_j = RANDOM\_UNIFORM(F(j))$, as well as compute objective function f(x) and measure of infeasibility v(x) for each solution. Set up the initial population P(0) ordering the generated solution by rule BEST-WORST so that the first solution (solution No.1) in population is the best one and the last solution (solution No.M) is the worst one.

STEP 2.

a) If the number of generated offspring is less than βK (optimization process stage

BREADTH) then choose one genetic operator from the set {RM, S1, S2, RX}, where selection probability of the operators are: $p_{RM}=\gamma$, p_{S1}, p_{S2}, $p_{RX}=1-\gamma-p_{S1}-p_{S2}$.

b) If the number of generated offspring is greater than βK (optimization process stage DEPTH) then choose one genetic operator from the set {LO, S1, S2, RX}, where selection probability are: $p_{LO}=\gamma$, p_{S1}, p_{S2}, $p_{RX}=1-\gamma-p_{S1}-p_{S2}$.

STEP 3.

For the chosen genetic operator sample, with uniform distribution, one or two parents (subject to the type of genetic operator: one for unary operator and two for crossover operator) from the old population P(t-1):

a) less TOPTAB best solutions (i.e., the solutions No.1,2,...,TOPTAB) if the number of generated offspring is less than βK;

b) less ENDTAB worst solutions (i.e., the solutions No.M-ENDTAB+1,M-ENDTAB+2,...,M) if the number of generated offspring is greater than βK;

STEP 4.

Using chosen genetic operator, generate offspring x^j, $j\in\{1,2\}$, and set up new population P(t): if the offspring x^j is better than the worst solution in population P(t-1), then solution x_{worst} is removed from the population and offspring x^j is introduced between other population solutions in accordance with ordering rule BEST-WORST.

STEP 5.

After generation K offspring return x_{approx}= arg max { f(x): x∈P(t) and v(x) is minimal }.

Let us notice that the population size M, total iteration number K and coefficient β, tabu numbers TOPTAB and ENDTAB and genetic operator selection probabilities γ, p_{S1}, p_{S2}, p_{RX} are GENGGP3 algorithm parameters; RANDOM_UNIFORM procedure is sampling procedure with uniform distribution.

4. Computer experiments

The basic genetic algorithm's fault, when an optimization problem is considered, is inability to represent the problem constraints like size constraint (3). In our computer experiments we consider two ordering rules BEST-WORST for the population solutions that allow constraint optimization problems to be addressed.

The first ordering rule uses penalty function formulation. In this case the evaluation (i.e., fitness) function EV(x) of the GGP problem solution x consists of the objective function f(x) together with function v(x) (see, formula (5)):

$$EV(x)= f(x)+w*v(x) \qquad (6)$$

where w, $w\geq 0$, is the weight coefficient of the evaluation function. Let us notice that coefficient w defines the part of infeasibility measure of the solution in evaluation function. There is no accepted methodology for choosing the weight coefficient w except rather large number of computer experiments. In case of incorporating a high penalty coefficient into the evaluation function we take the risk that if feasible solution is found, it drives the others out and the population converges on it without finding better solutions, since the likely paths to other feasible solutions require the production of infeasible solutions as intermediate structures, and the penalties for violating the constraint make it unlikely that such intermediate structures will reproduce. On the other hand, if the penalty coefficient is rather small the algorithm, especially for the heavily constrained problems, may generate solutions that violate the constraints.

In this modification function EV(x) is used for linear ordering of the set P solutions: the first (i.e., the best) solution of the population is that one with the maximal EV(x) value:

$x_{best} = x_{first} = \arg\max\{EV(x): x \in P\}$,..., the last (the solution No.M) is the worst solution $x_{worst} = x_{last} = \arg min\{EV(x): x \in P\}$.

The second ordering rule uses special case of Pareto formulation. Solutions in the set P are mapped into Euclidean two-space. The first coordinate of the image is the function v(x) value and second coordinate is the objective function f(x) value. Non-dominated solutions are assigned to the first Pareto front. The remaining nondominated solutions form the second front and so forth. Each Pareto front is additionally ordered in accordance with the function v(x) value: the first (the best) solution of the front is that one with minimal v(x) value and the last (the worst) one is that with maximal v(x) value. The linearly ordered Pareto fronts determine the linear order in population set P: the first solution of the first ordered Pareto fronts defines the best (the first) solution x_{best},..., the last solution in the last ordered Pareto front defines the worst (the last) solution x_{worst} in the population P.

On the basis of GENGGP3 algorithm the computer programming system, coded in C for IBM PC compatible computers under MS DOS operating system, was developed. In this section we report computer experiments realized with the aid of this programming systems for a test example of size n=30 and m=6, where the test data of matrices $[c_{jk}]_{30x30}$ and $[a_{ij}]_{6x30}$ are presented in papers [3],[4]. The matrix $[b_i]$ is equal to [17,18,18,15,13,16].

All computer experiments, performed by means of GENGGP3 algorithm, consisting of K = 15000 search process iterations for the following constant algorithm parameters: M=100, γ =0.1, $p_{S1}=p_{S2}=p_{RX}=0.3$. In tables below we use the following notations:

f(x_{approx}) - objective function value of the best solution x_{approx},

v(x_{approx}) - measure of infeasibility of solution x_{approx},

NL - number of search process iteration when the best solution x_{approx} was found.

Table 1 presents results of twenty experiments testing the first ordering rule BEST-WORST for following algorithm parameters: TOPTAB=5, ENDTAB=50, $w \in \{50,100,...,20000\}$, $\beta \in \{0.33, 0.67\}$. Let us note that in this case we have to perform some number of experiments to determine the right weight coefficient w.

Table 1. Experiment results for first ordering rule

	β=0.33			β=0.67		
w	$f(x_{approx})$	$v(x_{approx})$	NL	$f(x_{approx})$	$v(x_{approx})$	NL
50	6892	-13	176	6892	-13	176
100	8374	-11	814	8374	-11	814
200	9486	0	3521	9486	0	3521
300	11264	0	8053	11264	0	12645
400	11264	0	12365	11264	0	11295
500	11180	0	8599	11264	0	10027
1000	11180	0	11293	11066	0	12464
5000	11264	0	9097	11142	0	11647
10000	11188	0	5683	11172	0	13788
20000	11180	0	6025	11116	0	13615

The results of twenty experiments testing the second ordering rule BEST-WORST are presented in Table 2, where: TOPTAB=5, ENDTAB=50, $\beta \in \{0.33, 0.67\}$.

Table 2. Experiment results for second ordering rule

Exper., No	β=0.33			β=0.67		
	$f(x_{approx})$	$v(x_{approx})$	*NL*	$f(x_{approx})$	$v(x_{approx})$	*NL*
1	11.178	0	8.609	11.178	0	11.785
2	11.264	0	8.854	10.592	0	14.626
3	11.140	0	12.354	11.002	0	10.008
4	11.188	0	7.320	11.142	0	12.530
5	11.264	0	11.543	11.140	0	12.269
6	11.178	0	9.971	11.116	0	12.719
7	11.178	0	8.811	11.064	0	11.941
8	11.140	0	8.251	11.234	0	12.820
9	11.142	0	7.347	11.140	0	10.584
10	11.234	0	5.670	10.312	0	12.733
mean values	11.190,6	0	8.873	10.992	0	12.201,5

We did also experiments with other Pareto formulation. The first coordinate of the set P image into Euclidean two-space was the number of size constraints (3) violated instead of function v(x) value. In this case all search processes converge into solutions violated exactly one of m constraints (3) and the objective function was equal to the sum of all edge weights of the graph.

5. Conclusion

The paper describes the approximate algorithm GENGGP3 for the generalized graph partitioning problem. GENGGP3 algorithm is an instance of two stage genetic type algorithm with exploits five genetic operators, three of them perform simple search operation besides random choose operation. We have shown that these algorithms can be used on IBM PC compatible computers to solve successfully GGP problem in case of application special Pareto formulation, second ordering rule, for population solutions ordering.

References

1. Chen C. C.,(1986) Placement and partitioning methods for integrated circuit layout. Berkeley, CA, Ph. D. Dissertation, Dep. of EECS, Univ. of California.
2. Feo T. A., Khellaf M.,(1990) A class of bounded approximation algorithm for graph partitioning, Networks 20: 181-195.
3. Kadłuczka P., Wala K.,(1993) Tabu search heuristic for generalized graph partitioning problem, Automatics 64, 473-487, Cracow, University of Mining and Metallurgy Press.
4. Kadłuczka P., Wala K.,(1995) Tabu search nad genetic algorithms for generalized graph partitioning problem, Control and Cybernetics 24, 4, 1-18.
5. Kusiak A.,(1991) Group technology in flexible manufacturing systems, in Handbook of flexible manufacturing systems, Ed. N. K. Jha, Academic Press, Jnc.
6. Ma P.R., Lee E.Y.S., Tsuchiya M., (1982) A task allocation model for distributed computing systems, IEEE Transactions on Computers 31, 1, 41-47.
7. Michalewicz Z.,(1992) Genetic algorithms + data structures = evolution programs, Berlin, Springer - Verlag.
8. Wala K., Chmiel W.,(1995) A new genetic search scheme on an example of parallel machine scheduling problem, Silesia Technical Univ. Press, Automatics 116, 67-77, Gliwice, (in Polish).