Control and Cybernetics

vol. **24** (1995) No. 4

Tabu search and genetic algorithms for the generalized graph partitioning problem

by

Piotr Kadłuczka, Konrad Wala

Institute of Automatics,
University of Mining and Metallurgy
al. Mickiewicza 30, 30-059 Kraków, Poland

The paper considers the generalized problem of partitioning the 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. The applications of the investigated model for the group technology classification and for processing large computer programs in distributed computing system are presented.

Both tabu search and genetic algorithms are fine tuned for solving of the generalized graph partitioning problem. Tabu search algorithm is the neighbourhood search optimization procedure based on the tabu search framework with short as well as long term memory components, and a strategic oscillation element that allows search paths to cross the capacity-feasibility boundary. Genetic algorithm is an instance of genetic type algorithms with evaluation function consisting of objective function and weighted infeasibility measure. It exploits five genetic operators, three of them perform simple search operations besides random choose operations.

1. Introduction

The graph partitioning problem (GP) of an undirected and weighted graph $G = (N, E)$ consists of a partition of the nodes $N = \{1, 2, \ldots, j, \ldots, n\}$ into m disjoint subsets $X_1, X_2, \ldots, X_m$ so called clusters to optimize a given measure defined by the objective function $f(X)$, where $X = (X_1, X_2, \ldots, X_i, \ldots, X_m)$ is called the graph partition. This clustering of the nodes of a graph into subsets is NP-hard combinatorial problem (see, Feo and Khellaf (1988, 1990) for survey). Therefore there is a great demand for efficient approximation algorithms producing good suboptimal solution within a reasonable amount of time.

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, Chen (1986), in storing or processing large computer programs in distributed

computing systems, Ma, Lee and Tsuchiya (1982), Wala and Werewka (1989), and in group technology classification problems, Kumar, Kusiak and Vanelli (1986), Kusiak (1991).

Number of intractable combinatorial optimization problems can be solved now successfully by means of metaheuristics calling on principles of artificial intelligence. Machine learning becomes a popular field of research during the last years in artificial intelligence based on the idea to build computer programs capable to guide a search process into a "promising regions" for finding high-quality solution of an optimization problem. There has been an increasing interest in experimental comparisons of applying various new techniques to the same data sets of the same problems to provide a necessary condition to understand the merits of different methods. The most interesting machine learning approaches to hard combinatorial optimization problems are tabu search framework originally proposed by Glover and general purpose search strategies, proposed by Holland, called genetic algorithm.

The paper is organized in the following way: the next section presents the mathematical model of the generalized graph partitioning problem. In section 3 we look into details of two applications of the proposed model. Two approximate algorithms based on machine learning, i.e. tabu search algorithm TABGGP and genetic algorithm GENGGP, are presented in sections 4 and 5, respectively. These algorithms are fine tuned for solution of the generalized graph partitioning problem. The computer experiment results for a test example are reported in section 6. The last section contains conclusions.

2. Generalized graph partitioning problem

Mathematically, generalized graph partitioning problem (GGP) can be described 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 $i = 1, \ldots, m$. Find a graph partition $X = (X_1, X_2, \ldots, X_i, \ldots, X_m)$, $X_i \subset N$ $(i = 1, \ldots, m)$, of the set N which maximizes:

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

subject to:

$$\bigcup_{i=1}^{m} X_i = N \tag{2}$$

$$X_i \cap X_v = \emptyset \text{ for } i \neq v \tag{3}$$

$$\sum_{j \in X_i} a_{ij} \leq b_i, \; i = 1, \ldots, m \tag{4}$$

where:

$a_j = (a_{1j}, a_{2j}, \ldots, a_{ij}, \ldots, a_{mj})$ weight vector of the node $j \in N$,
c_{jk} weight of the edge $jk \in E$.
Constraints (2) and (3) ensure that each graph node is allocated precisely in one cluster (set) X_i. Constraints (4) limit the size for each cluster. The objective function $f(X)$ is the aggregate measure accounting for intra-cluster similarities or proximities (see, e.g. Owsiński (1984)). Let us note that partitioning the graph nodes, such that one minimizes the sum of the weight of the edges having incident nodes not in the same set X_i, is identical to clustering the nodes of the graph such that one maximizes the sum of the weights of the edges having incident nodes in the same set X_i, $i = 1, \ldots, m$. This follows from the fact that the sum of the weights of all edges in the graph is a constant and that each edge is either in a cluster or runs between two clusters. Thus the aggregate measure accounting for inter-cluster dissimilarities or distance is equal:

$$g(X) = \sum_{i=1}^{m} \sum_{v=1}^{m} \sum_{v \neq 1} \sum_{j \in X_i} \sum_{k \in X_v} c_{jk} = c - f(X) \tag{5}$$

where:

$$c = \sum_{j=1}^{n} \sum_{k=1}^{n} c_{jk} \tag{6}$$

is the constant value.

It is assumed, without loss of generality, that for each $jk \in E$: $c_{jk} > 0$ and for $jk \notin E$: $c_{jk} = 0$ as well as $a_{ij} = \infty$ if node j can not be assigned to cluster X_i, otherwise: $0 < a_{ij} < \infty$ for $i = 1, \ldots, m$; $j = 1, \ldots, n$. Let $F(j)$ denote the set of feasible cluster numbers for node j: $F(j) = \{i \in \{1, 2, \ldots, m\} : a_{ij} < \infty\}$, $j \in N$. Further, we will mark the feasible assignment of node j to cluster X_i, $j \in X_i$ and $i \in F(j)$, as a pair (i, j). 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), (3) and (4) the generalized graph partitioning (GGP) as opposed to standard GP where for each i: $a_{ij} = a_j$ and the constraint (4) has the form $\sum_{j \in X_i} a_j \leq b_i$.

3. Applications of the GGP model

Let us model the allocation problem of the distributed computer program into a distributed computing system (see e.g., Wala and Werewka (1989)). A distributed program consists of a set $N = \{1, \ldots, n\}$ of computational processes which cooperate to reach a common goal. The processes cooperate by exchanging messages and let c_{jk} stands for the number (or number per unit time) of message transmissions between the processes $j, k \in N$, where $c_{jk} = 0$ means that processes j and k do not cooperate and $E = \{jk \in N \times N : c_{jk} > 0\}$. The processes should be allocated over different processors to enable a parallel

computation. The solution of the allocation problem is given by the collection of the sets $X_1, X_2, \ldots, X_m$, where X_i, $X_i \subset N$, is a subset of processes allocated in processor i, for $i = 1, \ldots, m$. When the processes are allocated to the same processor (e.g. X_i processes) an information exchange is performed, in practically no time, using the memory of the processor. In the case the processes are in different processors the messages are transmitted through the communication channels and it may cause some overhead. One can maximize the message exchange by processors memory and use thus defined $f(X)$ as a objective function or minimize communication overhead using an other objective function $g(X)$. In this example of GGP a_{ij} stands for average processing time of the process j in processor i and b_i is the time resource of the processor i.

A second application of the GGP model arises in group technology classification problems. The basic idea of group technology is to decompose a manufacturing system into m subsystems called machine cells. The result of this grouping leads to physical machine layout. Some studies have shown that grouping machines into machines cells might limit the manufacturing system flexibility. However, industrial applications have proved that it is virtually impossible to implement a large-scale automated manufacturing system without using the cellular concept. For more information reader may refer to Kumar, Kusiak and Vannelli (1986), Kusiak (1991), Sawik (1992).

To model group technology problem GGP formulation can be used. In the graph $G = (N, E)$ the nodes set N represents the manufacturing system machines and subset X_i the machines allocated at the i-machine cell, $i = 1, \ldots, m$. Let the manufacturing system produce on the average μ_r workpieces (parts) of type r per time unit (week, month) and let $\delta_{jk}(r)$ denote a number of transfers of one workpiece from machine j to machine k during the production process realized according to the technological route of part type r. In this case we calculate $c'_{jk} = \sum_r \mu_r \delta_{jk}(r)$ and $c_{jk} = c'_{kj} + c'_{jk}$, thus the edges set $E = \{jk \in N \times N : \ c_{jk} > 0\}$. When the machines are allocated to the same cell the parts transport is performed using the cell robot. In the case the machines are in different cells the material handling carriers are used to transport workpieces, for example automated guided vehicles (AGVs). Each of the carriers serves a limited number of machines (e.g. maximum number of trips that an AGV can make per time unit is limited, see Kusiak (1991)). Thus one can maximize the workpiece transport by cell robot and use $f(X) \to \max$ as an objective function or minimize AGVs transport using the function $g(X) \to \min$. If given machines j and k have to be grouped together (due to technological requirements, for example, a forging machine and a heat treatment station, see Kusiak (1991)) one have to set $c_{jk} = A$, where A is a sufficiently large number.

In general, a manufacturing system may consist of various kind of cells thus resources b_i $(i = 1, \ldots, m)$ of these cells may be expressed in $[m]$, $[m^2]$, [machine post], [situation number] or [radian]. Figures 1, 2 and 3 illustrate these various kind of cells. In Fig. 1, the resource b_1 of the cell $i = 1$ is expressed in $[m]$, the machines can be allocated along the transport robot TR thus the

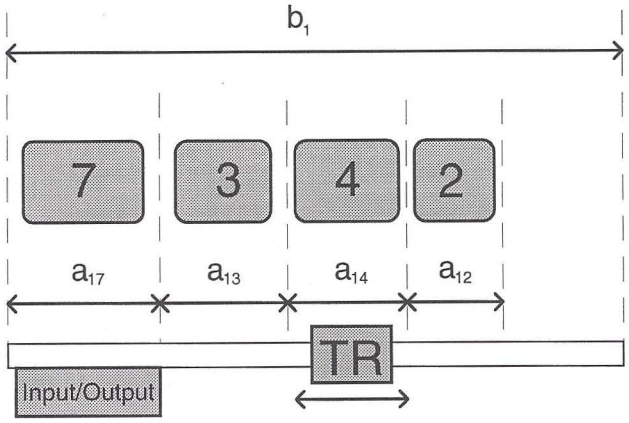

Figure 1. Cell $i = 1$ with transport robot TR (b_1 in $[m]$).

numbers a_{1j} $(j = 1, \ldots, m)$ are also expressed in $[m]$. In Fig. 2, the numbers b_2 and a_{2j} of the cell $i = 2$, with gantry robot GR, may be expressed in $[m^2]$ or [machine post] and for $i = 3$ (Fig. 3), b_3 and a_{3j} are expressed in radian, where $j = 1, \ldots, n$.

4. Approximation algorithm TABGGP

In our tabu search algorithm for solution the GGP, described by the objective function $f(X)$ and constraints (2), (3) and (4), we follow the tabu search algorithm developed for the multilevel generalized assignment problem in Laguna, Kelley, Gonzalez-Velarde, Glover (1991). We have designed a tabu structure that consists of dynamic tabu list with a long term memory component, and a strategic oscillation element that allows searching paths to cross the capacity – feasibility boundary.

Tabu search method derived by F. Glover exploits a collection of principles of intelligent problem solving. A fundamental element underlying tabu search is the use of flexible memory which embodies the dual processes of creating and exploiting structures for taking advantage of search history. In addition tabu search methods operate under the assumption that a neighbourhood $NB(X)$ of the current solution $X = (X_1, \ldots, X_m)$ can be constructed to identify "adjacent solutions" that can be reached from any current solution. Each solution $X' \in NB(X)$ can be reached directly from X by an operation called a *move*, and X is said to move to X', when such an operation is performed. There are used choice criteria for selecting "best" moves from $NB(X)$ and termination criteria for ending the search (for details see Glover (1989, 1990)).

In TABGGP algorithm the neighbourhood is defined as follows:

$NB(X) = N_1(X) \cup N_2(X)$, where

$N_1(X) = \{X'$: solution X move to solution X' consists in assignment replacement of node j from i_1-cluster to i_2-cluster, where $i_1 \neq i_2$, $i_2 \in F(j)\}$

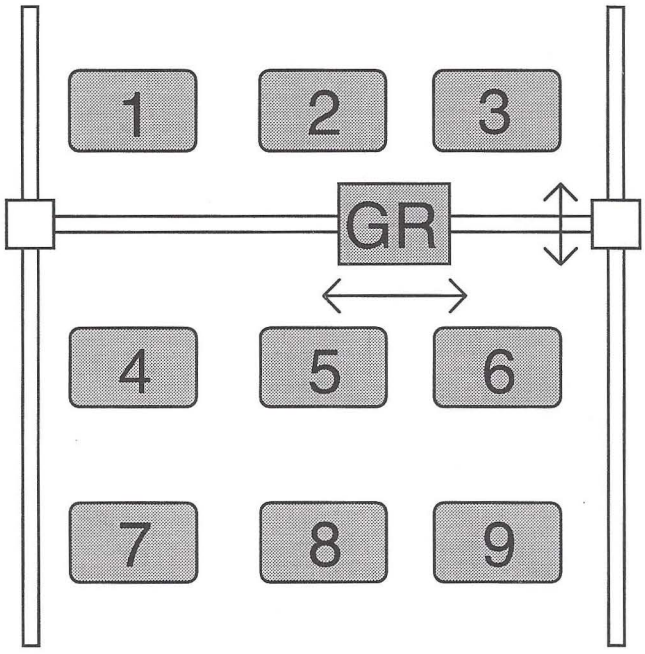

Figure 2. Cell $i = 2$ with gantry robot GR ($b_1 = 9$ machine post number).

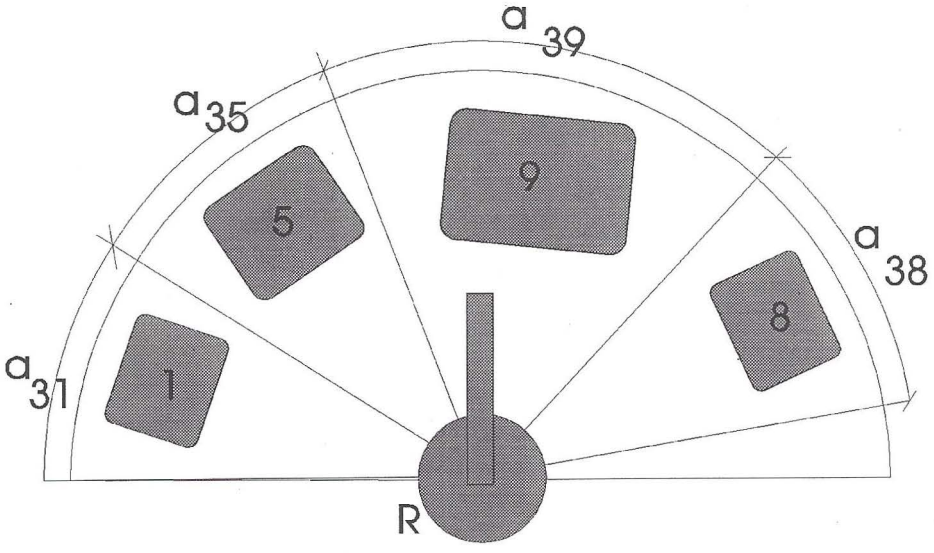

Figure 3. Cell $i = 3$ with robot R ($b_3 = \sqcap$ radian).

$N_1(X) = \{X'$: solution X move to solution X' consists in assignment replacement of node j_1 from i_1-cluster to i_2-cluster as well as node j_2 from i_2-cluster to i_3-cluster, where $j_1 \neq j_2$ and $i_1 \neq i_2$, $i_2 \neq i_3$ and $i_2 \in F(j_1)$, $i_3 \in F(j_2)\}$

The neighbourhood N_1 defines a very simple moves for GGP: a move is one that changes sets X_{i_1} and X_{i_2} for $X_{i_1} - \{j\}$ and $X_{i_2} \cup \{j\}$; i.e. only one assignment (i_1, j) changes for (i_2, j), $i_2 \in F(j)$. The neighbourhood N_2 is more complex: a move is one that changes sets X_{i_1}, X_{i_2} and X_{i_3} for $X_{i_1} - \{j_1\}$ and $\{X_{i_2} \cup \{j_1\}\} - \{j_2\}$ and $X_{i_3} \cup \{j_2\}$; i.e. two assignments (i_1, j_1) and (i_2, j_2) change for (i_2, j_1) and (i_3, j_2), $i_2 \in F(j_1)$, $i_3 \in F(j_2)$.

Let us define s_i, the capacity slack of i-clusters as follows. Let X be a given solution to the GGP that satisfies the assignment constraints (2) and (3) but does not necessarily meet the capacity restrictions (4), then

$$s_i = b_i - \sum_{j \in X_i} a_{ij} \tag{7}$$

TABGGP algorithm ensures that the assignment constraints (2), (3) are always satisfied; however, infeasibility may occur due to violation of the capacity constraints (4). 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 sum of all the negative capacity slacks:

$$v(X) = \sum_{i=1}^{m} \min\{s_i, 0\} \tag{8}$$

where, to be precise, the solution $X = (X_1, X_2, \ldots, X_i, \ldots, X_m)$ is an assignment – feasible one if the constraints (2) and (3) are satisfied and for each $j \in X_i$, $i = 1, 2, \ldots, m$, the pair (i, j) is a feasible assignment (satisfies the condition $i \in F(j)$).

The starting solution of the TABGGP algorithm is the solution X_{st} that results from assignment nodes of the set N to m clusters by means of greedy procedure with objective function $v(X)$.

A chief mechanism for exploiting memory in tabu search is to classify a subset $NT(X)$ of the moves in a neighbourhood $NB(X)$ as tabu. As a basis for preventing the search from repeating move combination tried in the recent past, potentially reversing the effects of previous moves by interchanges that might return to previous position, we introduce tabu list $TL = [TL(i, j)]$ $(i = 1, \ldots, m;\ j = 1, \ldots, n)$. Thus, the subset of tabu-active moves can be defined as

$NT(X) = \{X' \in NB(X)$: solution X move to solution X' contains a assignment (i, j) change for which $TL(i, j) > 0\}$,

where:

$NB(X) = NT(X) \cup NF(X)$ and $NT(X) \cap NF(X) = \emptyset$,

$NF(X) = \{X' \in NB(X)$: no solution X move to solution X' contains a assignment (i,j) change if $TL(i,j) > 0\}$ defines a subset of tabu-inactive moves.

Let a component of the move is assignment replacement of the node j from i-cluster to i_1-cluster. We call the tuple (i,j) as a leaving assignment. After a move has been executed, the leaving assignment (or assignments, in the case of $N_2(X)$ neighbourhood) become(s) tabu. The number of iterations that a leaving assignment remains tabu is a function of three elements: (1) number mn; (2) the relative increment of the objective function; (3) the frequency that the leaving assignment has been a component of a previously selected moves. Then, the number of iterations that the leaving assignment (i,j) will remain tabu is given by the following expression:

$$TT(i,j) = INTEGER\left(\mu_1 mn + \mu_2 \frac{\Delta f}{f} + \mu_3 \frac{\phi_{ij}}{\phi_{\max}}\right) \tag{9}$$

where:

μ_1, μ_2, μ_3 = weight coefficients,

$\Delta f = \max\{0, f(X) - f(X')\}$, $f = f(X)$,

ϕ_{ij} = number of times leaving assignment (i,j) has been part of an executed move component,

$\phi_{\max}$ = maximum ϕ_{ij}, for all i, j.

The long term memory element of the tabu list is implemented in form of a frequency count. This component of tabu tenure is only a function of the history of the current search.

Algorithm: TABGGP

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

STEP 1. Determine the start solution X_{st} and set $X_{cur} := X_{st}$ { where X_{cur} is the current solution }.

STEP 2. Calculate $f_{cur} := f(X_{cur})$ and $v_{cur} := v(X_{cur})$.

If $v_{cur} = 0$ set $f_{approx} := f_{cur}$ and $X_{approx} := X_{cur}$, otherwise set $f_{cur} := 0$.

Set $TL(i,j) := 0$ for all i,j.

STEP 3. (a) For $v(X_{cur}) = 0$ do the following.

Search the $NF(X_{cur})$ neighbourhood of solution X_{cur} only to get the set solutions S:

$$S = \{X' : \ f(X') = \max\{f(X) : \ X \in NF(X_{cur})\}\}$$

and search the best solution in set S:

$$X_{best} := \arg\max\{v(X) : \ X \in S\}$$

(b) For $v(X_{cur}) < 0$ do the following.
Search the $NF(X_{cur})$ neighbourhood of solution X_{cur} only to get the set solutions S:

$$S = \{X' : \ v(X') = \max\{v(X) : \ X \in NF(X_{cur})\}\}$$

and search the best solution in set S:

$$X_{best} := \arg\max\{f(X) : \ X \in S\}$$

STEP 4. If $v(X_{best}) = 0$ and $f(X_{best}) > f_{approx} := f_{best}$, set $f_{approx} := f_{best}$, $X_{approx} := X_{best}$;

STEP 5. Search the $NT(X_{cur})$ neighbourhood of solution X_{cur} only to get the set solution ST:

$$ST = \{X \in NT(X_{cur}) : \ v(X) = 0\}$$

and if $ST \neq \emptyset$ search the best tabu solution in set ST:

$$X_{tab} = \arg\max\{f(X) : \ X \in ST\}$$

If $f(X_{tab}) > f_{approx}$ set $f_{approx} := f(X_{tab})$, $f_{best} := f(X_{tab})$, $X_{approx} := X_{tab}$, $X_{best} := X_{tab}$;

STEP 6. For all (i,j) with $TL(i,j) > 0$ set $TL(i,j) := TL(i,j) - 1$; and for all leaving assignment (k,l) of solution X_{cur} move to X_{best} set $TL(k,l) := TT(k,l)$.
Set $f_{cur} := f(X_{best})$, $X_{cur} := X_{best}$;

STEP 7. Repeat steps 3, 4, 5 and 6 K times, where K is the TABGGP algorithm parameter.

STEP 8. Return f_{approx} and X_{approx}

Let us notice that in STEP 3 a strategic oscillation element that allows search path to cross the capacity – feasibility boundary is realized. In the tabu search methodology it is important to create best move definitions that depend on the search state. When the current solution X_{cur} is inside the feasible region, $v(X_{cur}) = 0$ (STEP 3(a)), the best neighbour is the one with the largest objective function value. When the current solution is infeasible, $v(X_{cur} < 0$ (STEP 3(b)), the best neighbour is the one that reduces infeasibility the most.

STEP 5 employs a simple type of aspiration criterion, consisting of removing a tabu classification from a trial move when the move yields a solution better then the best obtained so far, i.e. when $f(X_{tab}) > f_{approx}$, the tabu restriction can be overridden: $X_{approx} := X_{tab}$ and $X_{cur} := X_{tab}$.

5. Approximation algorithm GENGGP

Holland (1975) proposed a theoretical treatment of evolutionary adaptation. Artificial intelligence systems are starting to genetic algorithms as the basis for machine learning systems to make good decisions (see, e.g. Goldberg (1989), Michalewicz (1992)). The genetic algorithm consists of a population and of the genetic operators. The knowledge, accumulated during the solution process, is encoded as a list of solutions called population. The solution improvement

process is realized by use of genetic operators. Every genetic operator generates new solutions called offspring on the basis of old solutions called parents. Old solutions are picked out from the population, and then the new solutions replace, if they are better, the worse population solutions. As an evaluation function $EV(X)$ of the GGP problem solution X the objective function $f(X)$ together with function $v(X)$ (see, formula (8)) is used:

$$EV(X) = f(X) + wv(X) \tag{10}$$

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.

Taking into account genetic algorithm requirements concerning the solution form of the investigated problem, we introduce the second form of the GGP problem solution $x = (x_1, x_2, \ldots, x_j, \ldots, x_n)$, 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, in this section we denote the GGP problem solution by symbol x corresponding to X. Genetic algorithm GENGGP exploits the following genetic operators:

1. Inversion operator
   Inversion operator generates one offspring on the basis of one parent x:
   a) choose one number j so that $1 \leq j < n$,
   b) invert the elements of the parent $x_1, x_2, \ldots, x_j$ after elements $x_{j+1}, x_{j+2}, \ldots, x_n$.

   If the parent is the solution of the framework $x = (x_1, x_2, \ldots, x_j, x_{j+1}, \ldots, x_n)$ then the offspring has a framework $x^1 = (x_{j+1}, \ldots, x_n, x_1, \ldots, x_j)$
2. Mutation operator 1
   Mutation operator 1 generates one offspring on the basis of one parent x:
   a) choose one number i so that $1 \leq i \leq n$ as well as the "best" number j, $j = \arg\max_{k \neq i}\{EV(x_1, x_2, \ldots, x_{i-1}, x_k, x_{i+1}, \ldots, x_{k-1}, x_i, x_{k+1}, \ldots, x_n) : k \in N\}$;
   b) swap element x_i of the parent for element x_j.

   If the parent is the solution of the framework $x = (x_1, x_2, \ldots, x_{i-1}, x_i, \ldots, x_{j-1}, x_j, \ldots, x_n)$ then the offspring has a framework $x^1 = (x_1, x_2, \ldots, x_{i-1}, x_j, \ldots, x_{j-1}, x_i, \ldots, x_n)$
3. Mutation operator 2
   Mutation operator 2 generates one offspring on the basis of one parent x:
   a) choose one number j, $1 \leq j \leq n$, as well as the "best" number i, such that

$$EV(x_1, x_2, \ldots, x_{j-1}, i, x_{j+1}, \ldots, x_n) = \max_{k \neq x_j}\{EV(x_1, x_2, \ldots, x_{j-1}, k, x_{j+1}, \ldots, x_n) : k \in F(j)\};$$

   b) replace element x_j of the parent with number i.

If the parent has the framework $x = (x_1, x_2, \ldots, x_{j-1}, x_j, \ldots, x_n)$ then the offspring has a framework $x^1 = (x_1, x_2, \ldots, x_{j-1}, i, \ldots, x_n)$

4. Crossover operator

   a) choose one number j, $1 \leq j < n$, and for two parents:

   $$x' = (x'_1, x'_2, \ldots, x'_j, x'_{j+1}, \ldots, x'_n)$$

   $$x'' = (x''_1, x''_2, \ldots, x''_j, x''_{j+1}, \ldots, x''_n)$$

   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
   Let $S(x) = \{x' : x' = (x_1, x_2, \ldots x_{j-1}, x'_j, x_{j+1}, \ldots, x_n) : x'_j \in F(j), x'_j \neq x_j, j \in N\}$ denotes the neighbourhood of the solution $x = (x_1, x_2, \ldots, x_j, \ldots, x_n)$.
   Local optimization operator generates one offspring x^1 on the basis of one parent x:

   a) determine the neighbourhood $S(x)$,

   b) find a solution $x' = \arg\max\{EV(\hat{x}) : \hat{x} \in S(x), \hat{x} \neq x\}$

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

Each genetic operator has the operation "choose" which is arbitrary choice. Besides, genetic operators mutation 1 and mutation 2 have choose operation together with simple search operation (choose the "best" number j or i). The crossover operator makes possible to sample new regions (see, Holland (1975)), whereas the local optimization operator is used to do the thorough search in one particular region of the search space. In the GENGPP algorithm, the chosen operations are realized by sampling mechanism and in this way random changes during the offspring generation process are made.

Algorithm: GENGGP

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

STEP 1 Randomly generate an initial population of 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 and save the evaluation function value for each solution.

STEP 2 Choose one genetic operator, where p_1 is the selection probability of inversion operator, p_2 is the selection probability of mutation operator 1, etc. (where $p_1, \ldots, p_5 \geq 0$, $p_1 + p_2 + p_3 + p_4 + p_5 = 1$), and choose, from the population, one or two parents.

STEP 3 Using chosen genetic operator, generate offspring x^j ($j \in \{1,2\}$). If the solution-offspring x^j is better than the worst solution in population, then the last one is replaced by x^j, i.e. if $EV(x^j) > EV(x_{worst})$, where $x_{worst} = \arg\min\{EV(x) : x \in \text{population}\}$, then replace solution x_{worst} in population by x^j.

STEP 4 Repeat from step 2 to step 4 K times.

STEP 5 Return x_{approx}, where $x_{approx} = \arg\max\{f(x) : x \in \text{population and } v(x) \text{ is minimal}\}$.

Let us notice that the population size M, total iteration number K, weight coefficient w and genetic operator selection probabilities $p_1, \ldots, p_5$ are GENGGP algorithm parameters. The structure of GENGGP algorithm is similar to modGA algorithm, where $r \in \{1,2\}$ (see, Chapter 4 in Michalewicz (1992)), and RANDOM_UNIFORM procedure is sampling procedure with uniform distribution.

6. A test example

On the base of TABGGP and GENGGP algorithms two computer programming systems, coded in C for IBM PC compatible computers under MS DOS operating system, were 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 $m = 6$. We applied random problem generator, called A (see, Kadłuczka, Wala (1993)), to create random instance using the integer uniform distribution function:

$$
\begin{aligned}
c'_{jk} &= \begin{cases} 0, & \text{for } j = k \\ RANDOM\_UNIFORM[RANDOM\_UNIFORM[0,99],0], & \text{otherwise} \end{cases} \\
c_{jk} &= c'_{jk} + c'_{kj},\ j,k = 1.2.\ldots,n \\
a'_{ij} &= RANDOM\_UNIFORM[0,9] \\
a_{ij} &= \begin{cases} a'_{ij} & \text{for } a'_{ij} > 0 \\ \infty & \text{for } a'_{ij} = 0 \end{cases} \quad i = 1,2,\ldots,m;\ j = 1,2,\ldots,n \\
b_i &= INTEGER\left(1 + 0.7 \cdot \sum_{j=1}^{n} a'_{ij}/m\right),\ i = 1,2,\ldots,m.
\end{aligned}
$$

The table 1 gives the matrix $[c_{jk}]$ and table 2 presents the matrix $[a_{ij}]$. The matrix $[b_i]$ is equal to [17,18,18,15,13,16]. In tables 3 and 4 we use the following notations:

$f_k(x_{approx})$ – objective function value of the best solution x_{approx} in the k-th experiment, $k \in \{1,2\}$,

$v_k(x_{approx})$ – measure of infeasibility of solution x_{approx} in the k-th experiment,

$NI_k(CT_k)$ – number of search process iteration (computation time of the PC 486DX2/50 MHz in [seconds]) when the best solution x_{approx}, in the k-th experiment, was found,

TAV_k –average tabu time tenure of one computational test in the k-th experiment.

Table 3 presents results of two experiments, performed by means of TABGGP algorithm, consisting of $K = 10.000$ search process iterations for weight coefficients $\mu_1 = 0.01$, $\mu_2 = 1, 50, 100$ and $\mu_3 = 1, 10, 100$. In the first experiment, contrary to the second one, the aspiration criterion was neglected: STEP 5 of the algorithm was inactive. These two experiments enable us to observe the influence of aspiration criterion and coefficients μ_2, μ_3 on the search process.

Let us notice that the tabu structure defined by expression (9) may be viewed as a form of multiple lists of dynamic size. The first two components of this expression implement the short term memory of tabu status and the coefficients μ_1, μ_2 determine the number of iterations that leaving assignments will be not allowed to be part of the solutions to prevent the algorithm to oscillate between local optimum solutions. The purpose of the short term memory is not to rule out cycling completely since this would in general result in heavy bookkeeping and loss of flexibility, but at least to make it improbable. This problem, by choosing coefficients μ_1, μ_2, is generally to be resolved by experimentation. The third component of expression (9), along with coefficient μ_3, implements the long term memory of tabu status in form of the frequency count function. This function records the leaving assignments of the best solutions found in some phase of the algorithm. In a subsequent phase, tabu search can then be restricted to the subset of neighbourhood and the scale of neighbourhood reduction is controlled by coefficient μ_3. This enforces what Glover calls a "regional intensification" of the search in "promising regions".

Table 4 presents results of two experiments, performed by means of GENGGP algorithm, consisting of $K = 20000$ search process iterations for population size $M = 50$ and weight coefficient w from the interval $[50, 20000]$. The selection probabilities of the first experiment equals $p_1 = p_2 = p_3 = p_4 = 0.25$, $p_5 = 0.0$ and of the second experiment are as follows $p_1 = p_2 = p_3 = p_4 = 0.23$, $p_5 = 0.08$. In these experiments one can observe the influence of the local optimization operator and coefficient w on the search process.

There is no accepted methodology for choosing the penalty coefficient w except computer experiments. Let us notice that 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,

j\k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	0	230	0	196	364	0	34	202	0	244	48	0	82	92	216	110	44	0	188	46	0	48	0	0	240	52	178	16	196	146
2	230	0	12	108	144	196	104	0	0	120	36	194	108	84	0	102	0	0	0	0	0	138	0	0	16	10	4	66	296	48
3	0	12	0	0	0	0	96	64	4	146	182	236	120	144	212	224	62	96	12	104	170	0	56	30	96	54	0	68	0	24
4	196	108	0	0	314	180	76	124	0	6	198	132	188	106	46	150	38	0	280	4	64	0	168	58	316	138	0	80	166	0
5	364	144	0	314	0	112	0	198	206	88	180	0	0	0	180	162	24	0	174	60	0	40	0	122	186	282	8	0	100	172
6	0	196	0	180	112	0	20	48	266	104	216	28	72	0	102	118	196	236	132	0	252	0	0	298	168	98	0	38	162	20
7	34	104	96	76	0	20	0	126	102	164	0	0	96	50	0	0		226	216	218	180	86	18	136	290	182	0	0	0	174
8	202	0	64	124	198	48	126	0	90	172	184	0	64	10	166	58	94	26	190	56	20	42	174	16	0	58	0	114	68	0
9	0	0	4	0	206	266	102	90	0	76	48	190	366	0	0	58	0	44	174	312	0	42	32	0	198	110	154	54	24	270
10	244	120	146	6	88	104	164	172	76	0	0	216	58	218	0	0	10	268	100	20	22	0	0	0	88	108	168	152	154	94
11	48	36	182	198	180	216	0	184	48	0	0	270	94	106	34	118	262	136	50	0	172	76	12	154	282	4	0	0	72	0
12	0	194	236	132	0	28	0	0	190	216	270	0	156	118	108	302	164	0	0	218	12	168	206	58	136	198	252	256	24	46
13	82	108	120	188	0	72	96	64	366	58	94	156	0	26	174	44	0	122	0	100	82	272	0	44	86	40	0	168	0	0
14	92	84	144	106	0	0	50	10	0	218	106	118	26	0	316	0	0	168	48	66	296	18	224	178	0	0	120	56	306	0
15	216	0	212	46	180	102	0	166	0	0	34	108	174	316	0	102	180	80	192	162	48	144	146	190	98	116	0	146	0	34
16	110	102	224	150	162	118	0	58	58	0	118	302	44	0	102	0	0	0	64	198	286	132	48	0	296	178	188	316	50	0
17	44	0	62	38	24	196	0	94	0	10	262	164	0	0	180	0	0	76	190	212	148	0	334	0	280	0	38	0	184	0
18	0	0	96	0	0	236	226	26	44	268	136	0	122	168	80	0	76	0	0	130	172	304	0	102	180	308	176	128	184	70
19	188	0	12	280	174	132	216	190	174	100	50	0	0	48	192	64	190	0	0	10	148	24	148	110	60	0	0	104	84	28
20	46	0	104	4	60	0	218	56	312	20	0	218	100	66	162	198	212	130	10	0	8	314	240	0	0	20	170	272	8	14
21	0	0	170	64	0	252	180	20	0	22	172	12	82	296	48	286	148	172	148	8	0	174	222	0	212	196	6	78	324	0
22	48	138	0	0	40	0	86	42	42	0	76	168	272	18	144	132	0	304	24	314	174	0	124	158	12	0	82	92	84	148
23	0	0	56	168	0	0	18	174	32	0	12	206	0	224	146	48	334	0	148	240	222	124	0	356	48	138	104	0	42	0
24	0	0	30	58	122	298	136	16	0	0	154	58	44	178	190	0	0	102	110	0	0	158	356	0	0	42	0	168	0	32
25	240	16	96	316	186	168	290	0	198	88	282	136	86	0	98	296	280	180	60	0	212	12	48	0	0	96	142	52	158	140
26	52	10	54	138	282	98	182	58	110	108	4	198	40	0	116	178	0	308	0	20	196	0	138	42	96	0	0	0	266	0
27	178	4	0	0	8	0	0	0	154	168	0	252	0	120	0	188	38	176	0	170	6	82	104	0	142	0	0	156	78	44
28	16	66	68	80	0	38	0	114	54	152	0	256	168	56	146	316	0	128	104	272	78	92	0	168	52	0	156	0	166	0
29	196	296	0	166	100	162	0	68	24	154	72	24	0	306	0	50	184	184	84	8	324	84	42	0	158	266	78	166	0	100
30	146	48	24	0	172	20	174	0	270	94	0	46	0	0	34	0	0	70	28	14	0	148	0	32	140	0	44	0	100	0

Table 1. Matrix $[c_{jk}]$

i\j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	∞	5	1	6	9	6	1	8	5	4	1	4	∞	1	9	4	6	6	5	8	1	9	2	7	9	7	∞	6	8	6
2	∞	1	6	7	8	5	6	2	6	3	8	∞	∞	1	7	5	8	7	1	6	3	9	6	3	6	9	9	7	7	5
3	3	7	5	8	9	5	4	1	6	3	2	4	2	2	4	5	8	2	6	7	8	7	4	6	6	4	4	5	9	6
4	∞	9	∞	9	8	7	4	4	∞	5	3	9	9	2	5	2	∞	2	∞	7	8	5	1	1	7	5	2	7	4	2
5	3	2	1	2	4	1	5	6	7	6	4	∞	3	7	∞	1	5	2	9	5	6	2	5	1	6	3	5	4	3	3
6	2	4	8	4	6	4	3	1	2	2	2	4	4	8	1	7	5	3	2	5	9	7	3	3	6	9	4	∞	9	8

Table 2. Matrix $[a_{ij}]$

Parameters		k=1			k=2		
		μ_3=1	μ_3=10	μ_3=100	μ_3=1	μ_3=10	μ_3=100
$NI_k(CT_k)$	μ_2=1	153, (140)	794, (729)	17, (15)	286, (263)	6060, (5657)	23, (21)
$f_k(x_{approx})$		11234	11264	8988	11264	11264	10488
TAV_k		2	5.26	90.64	1.98	5.49	87.86
$NI_k(CT_k)$	μ_2=50	2194, (2029)	5021, (4678)	17, (15)	416, (386)	6321, (5901)	23, (21)
$f_k(x_{approx})$		11264	11264	8988	11264	11264	10488
TAV_k		3.73	7.55	93.45	3.60	7.38	90.32
$NI_k(CT_k)$	μ_2=100	2130, (1969)	1308, (1204)	17, (15)	3290, (3053)	7510, (7005)	23, (21)
$f_k(x_{approx})$		11234	11010	8988	11264	11264	10488
TAV_k		5.64	9.81	94.68	5.63	9.42	90.74

Table 3.

w	$f_1(x_{approx})$	$v_1(x_{approx})$	$NI_1(CT_1)$	$f_2(x_{approx})$	$v_2(x_{approx})$	$NI_2(CT_2)$
50	6532	-19	42, (10)	7040	-20	62, (2)
100	7420	-7	206, (6)	5366	-19	72, (2)
200	7904	-3	987, (29)	9362	0	172, (5)
300	11264	0	7508, (100)	11264	0	1434, (35)
400	11264	0	9278, (119)	11264	0	670, (19)
500	11186	0	5298, (76)	11264	0	3314, (55)
1000	10828	0	10606, (131)	11066	0	11652, (142)
5000	11264	0	15397, (182)	11264	0	11988, (146)
10000	11188	0	8601, (111)	11264	0	1763, (39)
20000	11178	0	18554, (216)	11264	0	5936, (83)

Table 4.

may generate solutions that violate the constraints.

For the best objective function value $f(X) = 11264$, the respective solution $X = (X_1, X_2, \ldots, X_6)$ has the following clusters $X_1 = \{3, 11, 12, 14, 17, 21, 23\}$, $X_2 = \{28\}$, $X_3 = \{13, 18, 20, 22\}$, $X_4 = \{7, 25, 27, 30\}$, $X_5 = \{2, 4, 6, 16, 24, 26, 29\}$, $X_6 = \{1, 5, 8, 9, 10, 15, 19\}$.

7. Conclusions

The paper describes two approximate algorithms, TABGGP and GENGGP, for the generalized graph partitioning problem. TABGGP algorithm is the neighbourhood search optimization procedure based on the tabu search framework with short as well as long term memory components, and a strategic oscillation element that allows search paths to cross the capacity – feasibility boundary. GENGGP algorithm refers to genetic type algorithms with evaluation function consisted of objective function and weighted infeasibility measure. GENGGP exploits five genetic operators, three of them perform simple search operation besides random choose operations.

We have shown that these algorithms can be used, on IBM PC compatible computers, to solve GGP problem successfully but GENGGP algorithm is computationally more efficient. Let us notice that one iteration of TABGGP algorithm consists of time consuming complete search in current solution neighbourhood thus, in our investigation, one computational test performed by means of TABGGP algorithm lasted over twenty times than one performed with the aid of GENGGP.

References

CHEN C. C. (1986) Placement and partitioning methods for integrated circuit layout. Berkeley, CA, Ph. D. Dissertation, Dep. of EECS, Univ. of California.

FEO T. A., KHELLAF M. (1988) The complexity of graph partitioning. Manuscript, Operations Research Group, Department of Mechanical Engineering, Univ. of Texas at Austin.

FEO T. A., KHELLAF M. (1990) A class of bounded approximation algorithm for graph partitioning, *Networks* **20**, 181-195.

GLOVER F. (1989) Tabu search, part I. *ORSA J. on Computing* **1**, 3, 190-206.

GLOVER F. (1990) Tabu search, part II. *ORSA J. on Computing* **2**, 1, 4-32.

GOLDBERG D.E. (1989) *Genetic algorithms in search, optimization and machine learning*, New York, Addison-Wesley.

HOLLAND J. (1975) *Adaptation in natural and artificial systems*, MI, Univ. of Michigan Press, Ann Arbor.

KADŁUCZKA P., WALA K. (1993) Tabu search heuristic for generalized graph partitioning problem, *Automatics* **64**, 473-487, Cracow, University of Mining and Metallurgy Press.

KUMAR K.R., KUSIAK A., VANNELLI A. (1986) Grouping parts and components in flexible manufacturing system, *Eur. J. Operations Res.* **24**, 387-397.

KUSIAK A. (1991) *Group technology in flexible manufacturing systems*, In: Handbook of Flexible Manufacturing Systems, Ed. N. K. Jha, Academic Press, Jnc.

LAGUNA M., KELLEY J.P., GONZALEZ-VELARDE J.L., GLOVER F. (1991) Tabu search methods for the multilevel generalized assignment problem, Manuscript, Graduate School of Business and Administration, University of Colorado at Boulder.

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.

MICHALEWICZ Z. (1992) *Genetic algorithms + data structures = evolution programs*, Berlin, Springer-Verlag.

OWSIŃSKI J. (1984) *On a quasi-objective global clustering method*, In: Data Analysis and Informatics, III, E. Diday et al., Amsterdam, North Holland.

SAWIK T. (1992) *Discrete optimization in flexible manufacturing systems*, Polish Scientific Publishers (in Polish).

WALA K., WEREWKA J. (1989) Allocation of computational processes in a multiprocessor system, *Archiwum Automatyki i Telemechaniki* **34** (1-2): 197-215.