Raport Badawczy

Research Report

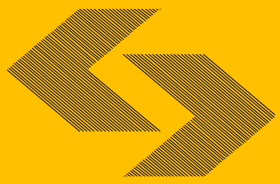
RB/79/2011

Evolutionary approach to obtain graph covering by densely connected subgraphs

J. Stańczak, H. Potrzebowski, K. Sęp

Instytut Badań Systemowych
Polska Akademia Nauk

Systems Research Institute
Polish Academy of Sciences



Control and Cybernetics

vol. **41** (2011) No. 3

Evolutionary approach to obtain graph covering by densely connected subgraphs*

by

Jarosław Stańczak, Henryk Potrzebowski and Krzysztof Sęp

Systems Research Institute Polish Academy of Science,
Newelska 6, Warsaw 01-447, Poland
e-mail: {stanczak, potrzeb, sep}@ibspan.waw.pl

Abstract: This article describes two evolutionary methods for dividing a graph into densely connected structures. The first method deals with the clustering problem, where the element order plays an important role. This formulation is very useful for a wide range of Decision Support System (DSS) applications. The proposed clustering method consists of two stages. The first is the stage of data matrix reorganization, using a specialized evolutionary algorithm. The second stage is the final clustering step and is performed using a simple clustering method (SCM).

The second described method deals with a completely new partitioning algorithm, based on the subgraph structure we call α-clique. The α-clique is a generalization of the clique concept with the introduction of parameter α, which imposes for all vertices of the subgraph the minimal percentage ($\alpha * 100\%$) of vertices of this subgraph that must be connected with vertices of this α-clique. Traditional clique is an instance of α-clique with $\alpha = 1$. Application of this parameter makes it possible to control the degree (or strength) of connections among vertices (nodes) of this subgraph structure. The evolutionary approach is proposed as a method that enables finding separate α-cliques that cover the set of graph vertices.

Keywords: graph, clique, graph clustering, evolutionary algorithms.

1. Introduction

Optimization of public communication or transportation systems, designing of nets and systems are only instances of a variety of possible applications of ideas described in this article. This domain is closely connected with detecting densely connected structures in graphs and can be called also graph clustering. Generally, graph clustering is a very wide domain with very rich literature for standard approaches (Falkner, Rendl and Wolkowicz, 1994; Jain and Dubes, 1988). The

*Submitted: July 2009; Accepted: June 2011.

methods of graph clustering, described in this article, are problem specific and rather prepared specially for problems depicted above, although they may be probably used in different domains.

A Design Structure Matrix (DSM, sometimes also Dependency Structure Method, Dependency Structure Matrix, Problem Solving Matrix (PSM), adjacency matrix, N-square matrix or Design Precedence Matrix) is a simple and compact representation of complex systems or projects, which can be decomposed and reorganized using some tools and methods. A DSM cell represents some process or processing unit and its interactions with other similar units. It is possible to track the information or goods flow by presentation of system relations analyzing the content of this matrix. This enables easy application of matrix-based methods that can improve the performance of considered systems.

This article presents a specialized evolutionary algorithm (EA) based method that helps to reorganize and detect strongly connected elements in the system structure, stored as a DSM. The EA changes the structure of the DSM changing the order of its rows and columns (permutations of rows and/or columns) to maximize a quality function, which is an equivalent of grouping strongly connected units. After the EA aided transformation of DSM, applying a simple clustering method enables extracting clusters of strongly connected units and finding a better structure of the system, taking into account the strength of connections among the elements of the projected/optimized system.

The second part of this article deals with the graph representation of optimized systems. The graph representation can be treated as a specific instance of DSM, but it has some additional features, which can be used for optimization purposes. In order to extract clusters of densely connected vertices in a graph one might cover it with separate cliques (disjoint complete subgraphs). But the classical clique seems to be in many cases a too strongly connected unit. In practical cases they may be very small and are difficult to tune to the requirements of a problem. Thus, we propose an extension of the clique notion, called α-clique. Introduction of parameter $\alpha \in (0, 1]$ enables to control the parameters of the clique, being the minimum proportion of vertices that must have connections with each other within their α-clique. So, tuning the parameter α gives the possibility to obtain different number and size of α-cliques or clusters that the whole graph is divided into.

The outline of this article is as follows. Some terms and symbols used in this article are defined in Section 2. In Section 3 evolutionary methods of graph partitioning are presented. Part 4 presents some computational results obtained using the described methods.

2. Basic concepts

2.1. The Design Structure Matrix (DSM)

Many decision problems deal with the task of appropriate design of a system structure, taking into account many interactions among its elements. The De-

sign Structure Matrix (DSM) is a good tool for representing and solving such problems (Browning, 2001).

The DSM contains elements a_{ij} (where $i \in \{1..|R|\}, j \in \{1..|S|\}, |.|$ denotes here the cardinality of a set, R and S are sets of elements of the system). The element a_{ij} is a measure of the relationship strength between elements of sets R and S. It can show some kind of connection between units (data or goods flow, communication connections, etc.). By proper permuting of rows and/or columns of such array it is possible to obtain groups, which are subsets of R strongly related to the corresponding subsets of S. The widely used method of DSM clustering is to maximize interactions among elements of the cluster while minimizing interactions among clusters (Lenstra, 1977).

The DSM clustering problem can be formulated as a well-known symmetric TSP (Traveling Salesman Problem) problem and the symmetric TSP problem can be formulated as a DSM clustering problem (Lenstra, 1977), which proves that it is a computationally hard problem and in practical cases only approximate methods can be used to solve it, for example: greedy, 2-opt, 3-opt (Lenstra, 1977; Sysło, Deo and Kowalik, 1983) or BEA, Bounded Energy Algorithm (McCormick, Schweitzer and White, 1972) or heuristic methods - genetic algorithms (Altus, Kroo and Gage, 1996; McCormick, Schweitzer and White, 1972; Potrzebowski, Stańczak and Sęp, 2004; Rogers, 1997; Yu, Goldberg, Yassine and Yassine, 2003; McCulley, Bloebaum, 1996). In our approach an adjusted evolutionary algorithm has been used to solve the clustering problem with the aid of a simple clustering method described in Section 3.2.

The adjustment of the genetic algorithm to the solved problem requires a proper encoding of individuals, inventing of specialized genetic operators and designing a fitness function to be optimized by the algorithm. The respective quality function is closely connected with the fitness function that values the members of the population. The classification problem is not a typical optimization task and its quality function is some artificial formula tuned to the problem. There are probably many possible fitness functions that can be used there.

The accepted solution (population member) encoding method requires specialized genetic operators, which modify the population of solutions. Simple random operators are similar to the widely used mutation and crossover (or exchange of parts of solutions). Also a set of heuristic operators was worked out and successfully tested: 2-opt and intelligent exchange.

The evolutionary method with the simple clustering algorithm and obtained results are discussed in the following sections.

2.2. Graphs

Notions described below are based on Wilson (1996).

A ***graph*** is a pair $G = (V, E)$, where V is a non-empty set of *vertices* and E is a set of edges. Each edge is a pair of vertices $\{v_1, v_2\}$ with $v_1 \neq v_2$.

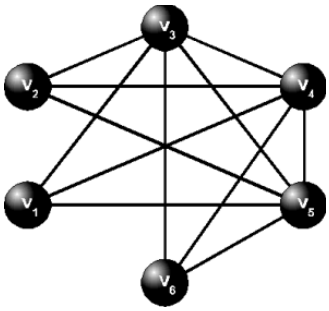


Figure 1. An example of a graph

In the graph from Fig. 1, $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and
$E = \{\{v_1, v_3\}, \{v_1, v_4\}, \{v_1, v_5\}, \{v_2, v_3\}, \{v_2, v_4\}, \{v_2, v_5\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_3, v_6\}\{v_4, v_5\}, \{v_4, v_6\}, \{v_5, v_6\}\}$.

Two vertices in the graph $G = (V, E)$ are called adjacent if for $v_i, v_j \in V$ there is $\{v_i, v_j\} \in E$.

For example: in graph from Fig. 1 vertices v_1 and v_3 are ***adjacent*** but vertices v_1 and v_2 are not adjacent

A ***subgraph*** of graph $G = (V, E)$ is a graph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$ such that for all $e \in E$ and $e = v_1, v_2$, if $v_1, v_2 \in V'$ then $e \in E'$.

A ***path*** in graph $G = (V, E)$ from vertex v_1 to vertex v_n is a sequence of vertices $\{v_1, ..., v_n\}$ such that $\{v_i, v_{i+1}\} \in E$ for $i = 1, 2, ..., n-1$ and for all v_i, v_j belonging to the path, if $i \neq j$ then $v_i \neq v_j$.

A ***degree*** of a vertex is the number of edges to which this vertex belongs.

There are following degrees of vertices in graph from Fig. 1 :
$\deg(v_1) = 3, \deg(v_2) = 3, \deg(v_3) = 5, \deg(v_4) = 5, \deg(v_5) = 5, \deg(v_6) = 3$.

Graph $G = (V, E)$ is a ***connected graph***, if for each pair of vertices there is a path between them.

Graph $G = (V, E)$ is a ***complete graph***, if for each pair of vertices there is an edge $e \in E$ between them, like in Fig. 2.

A ***clique*** (a *complete subgraph*) $Q = (V_q, E_q)$ in graph $G = (V, E)$ is a graph such that $V_q \subseteq V$ and $E_q \subseteq E$ and $Card(V_q) > 1$ or each pair of vertices $v_1, v_2 \in V_q$ fulfills the condition $\{v_1, v_2\} \in E_q$. If $Card(V_q) = 1$, Q is a clique.

For example: the graph in Fig. 2 is a clique. Each nonempty subgraph of a clique is a clique.

A ***maximum clique*** $Q_M = (V_q, E_q)$ in graph $G = (V, E)$ is a clique, for which there exists no vertex $v \in V$ and $v \notin V_q$ such that $Q' = (V', E')$ is a clique, where $V' = V \cup \{v\}$ and $E' \subseteq E$ and each pair $v_1, v_2 \in V'$ of vertices fulfills the condition $\{v_1, v_2\} \in E'$.

For example the entire graph presented in Fig. 2 constitutes the maximum clique but the graph shown in Fig. 1 does not. A maximum clique in graph from Fig. 1 compose the vertices v_3, v_4, v_5, v_6 with relevant edges.

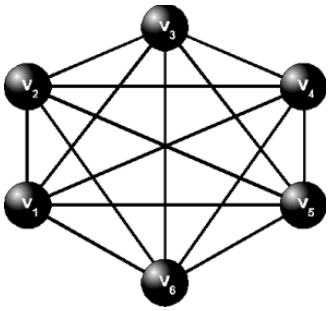


Figure 2. A complete graph

2.3. An α-clique

Let $A = (V', E')$ be a subgraph of graph $G = (V, E)$, $V' \subseteq V$, $E' \subseteq E$, $k = Card(V')$ and let k_i be a number of vertices $v_j \in V'$ such that $\{v_i, v_j\} \in E'$.

1. For $k = 1$ the subgraph A of graph G is an α-clique (α).
2. For $k > 1$ the subgraph A of graph G is an α-clique (α) if all vertices $v_i \in V'$ fulfill the condition $\alpha \leq \frac{k_i+1}{k}$, where $\alpha \in (0, 1]$.

Further on we will use the notion of α-clique in the sense of α-clique(α) for an earlier established α.

As it can be seen in Figs. 3 and 4, a subgraph of an α-clique(0.8) is not an α-clique(0.8), thus the property of being α-clique(α) may not be preserved by the sub-graphs of an α-clique.

Let α-clique $A = (V, E)$ be a graph with $\alpha > \frac{1}{2}$, thus, for all vertices v_i belonging to α-clique(α) $k_i + 1 > \frac{1}{2}k$.

The set theory implies the fact that for each pair of vertices, the sets of vertices adjacent to them have a non-empty intersection, so the α-clique(α) with $\alpha > \frac{1}{2}$ constitutes a connected graph.

If $\alpha > \frac{1}{2}$, the obtained subgraph may be disconnected - an example of such a situation is shown in Fig. 5.

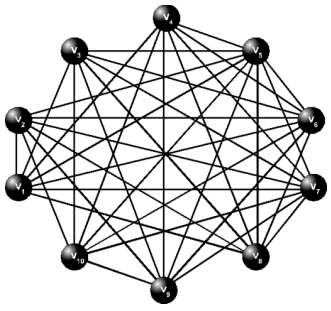


Figure 3. An example of α-clique(0.8)

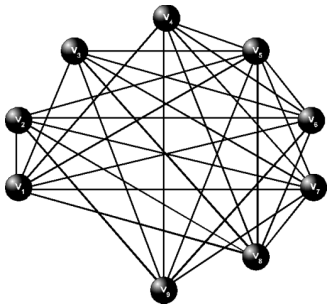


Figure 4. A subgraph of graph from Fig. 3 which is not an α-clique(0.8)

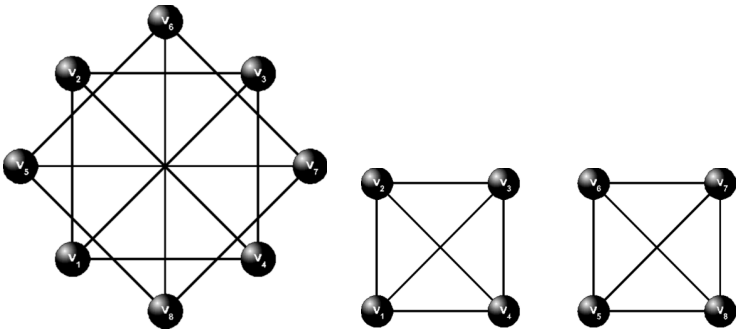


Figure 5. An example of a not connected graph for $\alpha > \frac{1}{2}$

It is useful to introduce the notion of boundary α_b:

$$\alpha_b = \min_i \left(\frac{k_i + 1}{k} \right).$$

There is another fact which should be emphasized. If a graph A is an α-clique(α_1) and $\alpha_1 \geq \alpha_2$, then A is also an α-clique(α_2). It means that A is an α-clique for every value of α less or equal α_b.

Generally, it seems handy to compute the value of α_b for each graph considered as an α-clique to know the maximum admissible value of α.

2.4. α-clique as a generalization of the notion of clique

As it has been mentioned, α-clique is a generalization of clique. Thus, it is expected that the maximum α-clique problem is more difficult that the maximum clique problem which is NPH (Aho, Hopcroft and Ullman, 1974; Bagirov and Yearwood, 2006; Benson and Ye, 2000; Aussiello et al., 1999; Cowen, 1998; Hansen, Mladenovic and Urosevic, 2004; Hassin and Khuller, 1986; Hochbaum, 1997; Hromkovic, 2001; Jukna, 2001; Korte and Vygen, 2000; Kumlander, 2007; Protasi, 2001). The maximum α-clique problem as a generalization of the maximum clique problem is NPH as well. This difficulty of finding α-cliques can be

observed taking into account the fact that each non-empty subset of a clique is also a clique, but this is not true in the case of α-clique. For an established value of α it is possible to find an α-clique(α) such that some of its subsets are not α-cliques(α) (Figs. 3 and 4). Thus, it is difficult to find almost maximal α-clique using simple greedy algorithms, which expand smaller ones by attaching new vertices to so far obtained solutions. Our method does not try to find maximal α-cliques, but optimizing the quality function (3) in exceptional circumstances can tend to this case and the algorithm that solves our problem must overcome similar difficulties like in the case of finding the maximal α-clique. Thus, we use the evolutionary algorithm to solve the problem, because this method can do it efficiently.

The exact algorithm

The exact algorithm bases on generating all nonempty subsets of the set of vertices. We propose lexicographical backtracking algorithm for obtaining the family of non-empty subsets(see an illustration in Fig. 6).

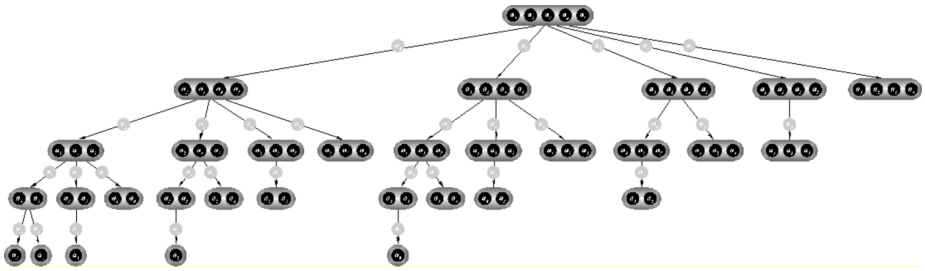


Figure 6. Backtracking tree for five vertices

Algorithm 1 - The exact algorithm

Input:
C set of elements
Output:
Z - set of family of all nonempty subsets of set C
Z=$\emptyset$;
Let A be a progression of all elements belonging to C.
Require: a_i - the first element to be removed from A, in the beginning $i = 0$
for all $a_j \in A$ such that $j \geq i$ **do**
begin
$Z = Z \cup A$
Call the procedure recursively with parameters $A := A - \{a_j\}, i := j$;
end;

This algorithm generates all non-empty subsets, thus for the n-element set the pessimistic complexity is $O(2^n)$.

3. Evolutionary algorithms to solve the graph clustering problem

3.1. Standard evolutionary algorithm

The standard evolutionary algorithm works in the manner shown in Algorithm 2, but this simple scheme requires many problem specific improvements to work efficiently (Michalewicz, 1996).

The adjustment of the genetic algorithm to the solved problem requires proper encoding of solutions, invention of specialized genetic operators proper for the accepted data structure and a fitness function to be optimized by the algorithm.

Algorithm 2 - The standard evolutionary algorithm

Input:
$G(V, E)$ input graph
Output:
Q set of $Q_i(V_i, E_i)$ - α-cliques of G such that $V = \cup_i V_i$ and $V_i \cap V_j = \oslash \; for \; i \neq j$
begin
Random initialization of the population of solutions.
while stop condition is not satisfied **do**
begin
Reproduction and modification of solutions using genetic operators
Valuation of obtained solutions
Selection of individuals for the next generation
end;
end;

Evolutionary algorithms are often used to solve difficult graph problems such as graph coloring, TSP, graph partitioning, maximum clique searching, etc. (Chen, Wang and Okazaki, 2008; Talbi and Bessiere, 1991; Marchiori, 1998), thus, it seems fully justified to use the evolutionary algorithm in graph clustering problem.

3.2. A DSM matrix method of clustering using an evolutionary algorithm and the simple clustering algorithm

3.2.1. Solution encoding in evolutionary algorithm

The whole information about the problem is stored the DSM matrix. There is only one global data table in the described approach. Members of the population (Fig. 7) contain their own solutions of the problem as vectors of row indices (symmetric case) or of rows and columns in the non-symmetric case. Vectors of indices are permutations of rows or columns of the single global data array. This method of encoding saves memory and reduces the computational complexity of the genetic operators, especially for big data arrays, compared to the easier

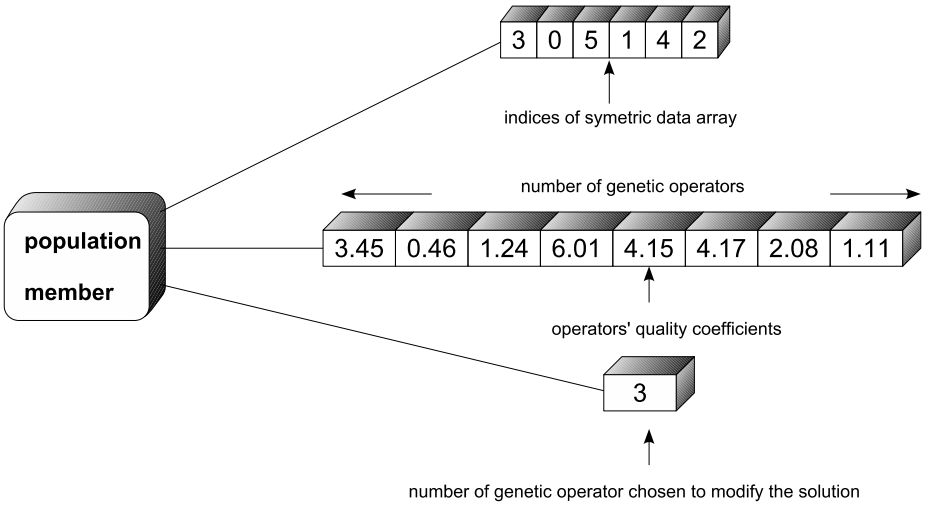


Figure 7. A structure of a population member (symmetric case)

method with separate DSM matrices stored in each population member. This approach complicates fitness function calculation, but the overall performance of this method is better.

Besides, a member of the population contains more data including: a vector of real numbers, which describes its knowledge about genetic operators, and the number of the chosen operator for the current iteration - more details about this data will be given later.

3.2.2. Fitness function

The quality function is closely connected with the fitness function, which evaluates the members of the population. In the problem of data clustering, the fitness function is almost identical with the problem quality function (1), only scaled to the interval (0,1),

$$Q = \frac{1}{2}\sum_{i=j}^{n}\sum_{j=1}^{m} a_{ij}(a_{i,j-1}+a_{i,j+1}+a_{i-1,j}+a_{i+1,j}) = \sum_{i=j}^{n}\sum_{j=1}^{m} a_{ij}(a_{i,j+1}+a_{i+1,j}) \quad (1)$$

where:
n, m – numbers of rows and columns, for symmetric case $n = m$; it is assumed that elements $a_{0,j}, a_{i,0}, a_{n+1,j}, a_{i,m+1}$ equal 0.
a_{ij} – element of the data array.

3.2.3. Specialized operators

The described data structure requires specialized genetic operators, which modify the population of solutions. Only operators permuting indices of rows and/or

columns are allowed in that problem:

- mutation: an exchange of randomly chosen subset of indices;
- multiple mutation: the mutation operator performed several times;
- intelligent exchange: one randomly selected index is exchanged with some others, solution with the best value of quality function is an effect of this operator;
- a multiple version of intelligent exchange is also applied;
- 2-opt operator: indices are exchanged in pairs, best found modification is stored as a new solution.

The difference between symmetric and non-symmetric case is that for the symmetric case operators work only on one vector of indices (indices of rows and columns are the same). In the non-symmetric case they work separately on the vector of indices to rows and on the vector of indices to columns.

3.2.4. Simple clustering method

After EA-aided preprocessing of DSM the following method is performed to obtain the final clustering. Let P be a value function for each row. For $j = 1, 2, \ldots$, determine

$$Pj = \sum_{i=j}^{n} a_{ij}(a_{i,j-1} + a_{i,j+1} + a_{i-1,j} + a_{i+1,j}) \tag{2}$$

A simple algorithm to explicitly divide a given order of units into clusters has the form of Algorithm 3.

This method produces some kind of histogram of the data matrix and enables detecting borders between highly connected areas. Especially good results can be obtained, when this method is used for final clustering after evolutionary preprocessing of data. This fact is illustrated in Fig. 8.

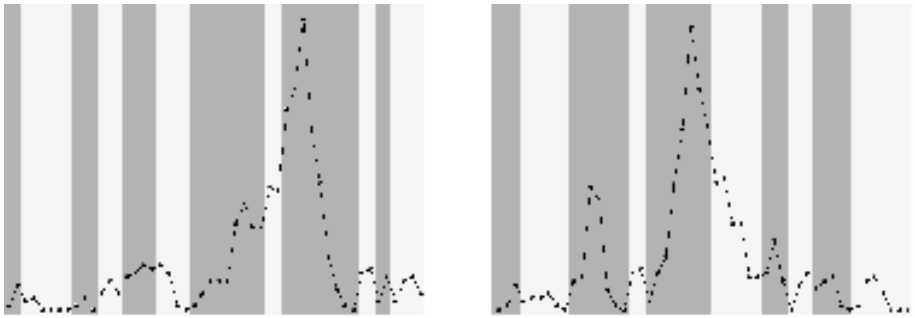


Figure 8. Results of clustering without (left, 12 clusters) and with EA preprocessing (right, 10 clusters)

Algorithm 3 - The SCM algorithm

```
Input:
        B- DSM matrix preprocessed by EA
        r - number of rows in the DSM matrix
Output:
        Q-Set of clusters
begin
        T: array [1 .. r+1];
        Q=∅;
        for i :=1 to r do
            T[i] : =P(B[i]);
        T[r+1] := ∞ ;
        j :=1;
        i := 2;
        while i <= r do
            begin
                if T[i-1] = T[i] do
                    begin
                        while T[i-1] = T[i] do
                            i := i+1;
                        if T[i-1] < T[i] do
                            begin
                                Q := Q ∪ j, i-1;
                                j := i;
                            end
                    end
                if T[i-1] > T[i] and T[i] < T[i+1] do
                    begin
                        Q := Q ∪ j, i-1;
                        j := i;
                    end
                i := i+1;
            end;
end;
```

3.3. The evolutionary approach to finding α-cliques

3.3.1. Representation of individuals

The whole information about the problem is stored in a square array of data describing all data connections. This array can be binary (adjacency matrix of undirected graph: 0 no connection, 1 presence of connection), non-negative (undirected graph) real-valued and in this case the stored value denotes the strength of the connection in a matrix with negative values (directed graph). This matrix can be also treated as an instance of DSM, but note that only square matrices can be used in this approach.

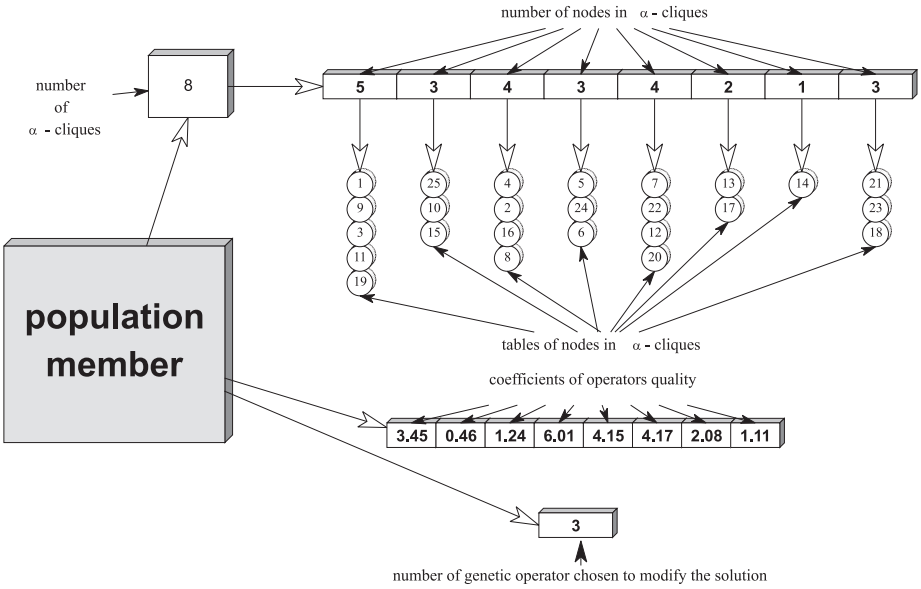


Figure 9. Structure of a population member

Members of the population (Fig. 9) contain their own solutions of the problem as a dynamic table of derived α-cliques (their number may change during computations). Each element of this table (α-clique) has a list of nodes attached to this α-clique and each node is considered only once in one solution (population member). Unattached nodes are also included, they constitute small, one-element α-cliques (one node is also an α-clique with α=1). Thus, each solution contains all nodes from a graph described by the adjacency matrix. But the solution with many small α-cliques is rather not advantageous, and it is the role of the evolutionary algorithm to find bigger ones. Besides it, a member of the population contains more data including: a vector of real numbers, which describe its knowledge about genetic operators and the number of the operator chosen to modify the solution in the current iteration. More details abut genetic operators and the method of their valuation will be given later on.

3.3.2. Fitness function

The quality function of the problem is closely connected with the fitness function, which evaluates the members of the population. In the solved problem several quality functions may be considered, depending on input data (binary, integer or real) or what kind of α-cliques one wants to obtain (equal size or maximal size etc.). The fitness function does not have to possess any penalty part for an α-clique constraint violation, because forbidden solutions are not produced

by population initializing function or genetic operators. Thus, all population members contain only valid α-cliques with their local values of α computed for all vertices of a subgraph not less than the global value imposed on the solved problem. For computer simulations we used the fitness function as follows:

$$\max Q = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{l_i} \sum_{k=1}^{j} D[t_{ik}, t_{ij}] \tag{3}$$

where:
n numbers of α-cliques in the solution;
l_i number of vertices in the i-th α-clique;
D the data array (adjacency matrix);
t_{ij}, t_{ik} vertices of the i-th α-clique.

The fitness function (3) promotes α-cliques of medium size and this version was used to solve the benchmark problems, but it is possible to propose different fitness functions (Potrzebowski, Stańczak and Sęp, 2006) with various properties of sizes of generated α-cliques, for example:

$$\max Q = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{l_i} \sum_{k=1}^{j} D[t_{ik}, t_{ij}] - \sum_{i=1}^{n} |l_i - \frac{l}{n}| \tag{4}$$

where:
l number of all vertices in the considered graph;
all other symbols like in the formula (3);
this function promoting possibly equal-sized α-cliques.

3.3.3. Specialized operators

The described data structure requires specialized genetic operators, which modify the population of solutions. Each operator is designed in such a manner that it preserves the property of being an α-clique(α) for the modified parts of solutions. If a modified solution violates the limitation of being an α-clique(α), the operation is canceled and no modification of the solution is performed. This makes it more difficult for the evolutionary algorithm to find satisfying solutions, due to possible bigger problems with local maximums, than the method with penalty function, but it gives the certainty that the computed solutions are feasible.

Genetic operators used are as follows:

1. mutation: an exchange of randomly chosen nodes in different α-cliques;
2. movement of randomly chosen node to a different α-clique;
3. intelligent movement performed only if this modification gives better value of fitness function;
4. concatenation: this operator tries to concatenate (mainly small) α-cliques;
5. also multiple versions of operators are applied.

3.4. Evolutionary algorithm used to solve the problem

The use of specialized genetic operators requires applying a selection method to execute them in all iterations of the algorithm. The traditional method with a small probability of mutation and a high probability of crossover is not applicable in this case, because there are more operators than two and their properties cannot be easily described as exploration or exploitation. In the used approach (Stańczak, 2003) it is assumed that an operator that generates good results should have higher probability and more frequently affect the population. But it is very likely that the operator, that is proper for one individual, gives worse effects for another, for instance because of its location in the domain of possible solutions. Thus, every individual may have its own preferences. Every individual has a vector of floating point numbers, besides the encoded solution. Each number corresponds to one genetic operation. It is a measure of quality of the genetic operator (a quality factor). The higher the factor, the higher the probability of using the operator. The ranking of qualities becomes a base to compute the probabilities of appearance and execution of the genetic operators. Simple normalization of the vector of quality coefficients turns it into a vector of operator execution probability (5). This set of probabilities can be treated as a base of experience of every individual and according to it, an operator is chosen in each epoch of the algorithm. Due to this experience one can maximize the chances of its offspring to survive,

$$p_{ij}(t) = \frac{q_{ij}(t)}{\sum_{i=1}^{L(t)} q_{ij}(t)} \tag{5}$$

where:
p_{ij} - represents probability of execution of the genetic operator,
q_{ij} - represents a quality factor of the genetic operator,
$L(t)$ - represents the number of genetic operators (in some evolutionary algorithms this number may vary during computations),
t - represents current time.

The method to compute the quality factors is based on reinforcement learning (Cichosz, 2000). An individual is treated as an agent whose role is to select and call one of the evolutionary operators. When the selected i-th operator is applied it can be regarded as an agent action a_i leading to a new state s_i, which, in this case, is a new solution. The agent receives reward or penalty respective to the quality of the new state (solution). The aim of the agent is to perform the actions which give the highest long term discounted cumulative reward V^*:

$$V^{\pi} = E_{\pi} \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{6}$$

$$V^* = \max_{\pi}(V^{\pi}) \tag{7}$$

where:
Π – is a strategy of the agent,
V^{π} – is a discounted cumulative reward obtained using strategy Π,
E – is an expected value,
γ – is a discount factor,
k – is a consecutive time steps,
t – is current time.

The following formula can be derived from (3) and (4) and is used for the evaluation purposes:

$$V(s_{t+1}) = V(s_t) + \alpha(r_{t+1} + \gamma V^*(s_{t+1}) - V(s_t)) \tag{8}$$

where:
$V(s_t)$ – a quality factor or discounted cumulative reward,
$V^*(s_{t+1})$ – estimated value of the best quality factor (in our experiments we take the value gained by the best operator),
α – a learning factor,
γ – a discount factor,
r_{t+1} – the reward for the best action, which is equal to the improvement of the quality of a solution after execution of the evolutionary operator,
t – current moment in time.

In the presented experiments the values of α and γ were set to 0.1 and 0.2, respectively.

Likewise, the applied method of selecting individuals to the new parent population (the selection method) has an important influence on the process of evolutionary computation. Commonly used methods, like proportional selection and tournament selection are often not the best choices. The applied selection method is an instance of controlled selection. That is, it can change its parameters according to the needs of the process. To obtain this, a mixed selection method was applied, composed of two methods with different selection properties: histogram selection (increases the diversity of the population) and deterministic proportional selection (strongly promotes best individuals) (Mulawka, Stańczak, 1999), chosen randomly during the execution of the evolutionary algorithm.

$$p_{his}(t+1) = \begin{cases} p_{his}(t) \cdot (1-a) & \text{for R(t)} > 3 \cdot \sigma(\text{F(t)}) \\ p_{his}(t) \cdot (1-a) + 0.5 \cdot a & \text{for R(t)} \in [0.5 \cdot \sigma(\text{F(t)}),\ 3 \cdot \sigma(\text{F(t)})] \\ p_{his}(t) \cdot (1-a) + a & \text{for R(t)} < 0.5\sigma(\text{F(t)}) \end{cases} \tag{9}$$

$$R(t) = \max(F_{av}(t) - F_{\min}(t), F_{\max}(t) - F_{av}(t))$$

where:
$p_{his}(t)$ – probability of histogram selection appearance in following iterations $(1 - p_{his}(t)$ is probability of deterministic roulette method $p_{det}(t))$;
$F_{av}(t)$, $F_{\min}(t)$, $F_{\max}(t)$ – average, minimal and maximal values of fitness function in the population;

$\sigma(F(t))$ – standard deviation of fitness function $(F(t))$ in the population of solutions;
a – a small value to change probability $p_{his}(t)$, in simulations $a = 0.05$.

If individuals in the population feature a too small standard deviation of the fitness function $(\sigma(F(t)))$ with respect to the extent of this function $(\max(F_{av}(t) - F_{\min}(t), F_{\max}(t) - F_{av}(t)))$, then it is desirable to increase the probability of the histogram selection. In the opposite case the probability of the deterministic roulette selection is increased. When parameters of the population are located in some range considered appropriate we may keep approximately the same probabilities of appearance for both methods of selection. It is important that always $p_{his}(t) + p_{det}(t) = 1$, which means that some method of selection must be executed.

4. Computer simulation results

4.1. The data used in simulations

4.1.1. Import-export connections of regions of Indonesia

The testing data set describes a square 50x50 problem with a non-symmetric matrix (data array), where 50 data attributes are considered. This problem is called "Import-export" example and has been taken from Benson, Ye (2000). It describes economical connections among 50 regions of Indonesia. In the source data $a_{ij} = 1$ if in 1971 a quantity of at least 50 tons of rice was transported from region i to region j, and $a_{ij} = 0$ otherwise. In the conducted simulations, beside the original data, we also considered a matrix converted to the symmetric version, where the direction of good flows was discarded and only the fact of trade was important (see Fig. 10). This was due to the fact that the source matrix is very sparse* and it is difficult to obtain results showing real trade connections, when only one direction is considered. But, of course, the presented methods are not limited to the symmetric cases and thus we present also clustering of the non-symmetric source data.

4.1.2. The problem of airport design

The data describe the problem of airport design. The data matrix contains 27x27 elements, which describe the strength of connections among the airport units. The data matrix A is symmetric, with weights ranging from 0 to 3. The shades of the entries in Fig. 11 denote the strength of connections between considered nodes (black-high strength of connection, white - no connection). The main aim of this task is to group elements in the matrix into highly connected areas which should be located closely to each other in the designed airport to

*Several vertices of this graph are isolated, thus in obtained results (Figs. 12 and 14) there are small clusters with only one vertex.

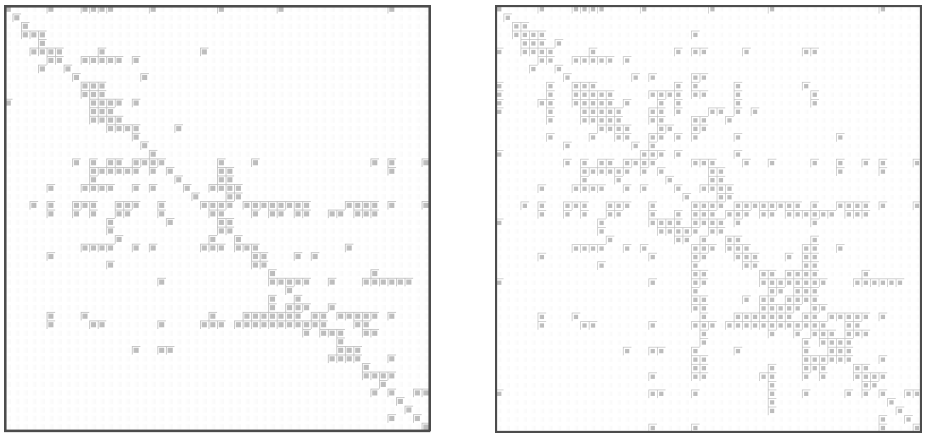


Figure 10. The source data for the import-export problem: before symmetrization (left) and after symmetrization (right)

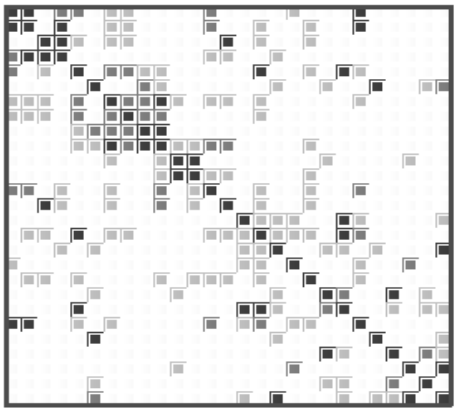


Figure 11. The source data for the airport design problem

obtain high flow capacity among them. The source data has been taken from (Browning, 2001; Lenstra, 1977).

4.1.3. Examples of large graphs used for testing the α-clique graph covering

The testing examples were taken from BHOSLIB: Benchmarks with Hidden Optimum Solutions for Graph Problems (Maximum Clique, Maximum Independent Set, Minimum Vertex Cover and Vertex Coloring) Hiding Exact Solutions in Random Graphs

http://www.nlsde.buaa.edu.cn/ kexu/benchmarks/graph-benchmarks.htm.

The first task was a rather large graph with 4000 vertices and 7 425 226 edges (file: frb100-40.clq.gz), the second problem was a smaller graph with 4000 vertices and 572 774 edges (file: frb100-40mis.gz).

4.2. Results of simulations

4.2.1. The import-export example of Indonesian regions

The aim of the analysis is to find trade connections among different regions of Indonesia. Solving this problem can help to find better model of connections among regions and to develop them into larger areas. The best solutions of the problem with explicit clustering are given in the following figures.

The DSM matrix method maximizes the quality criterion (1), aimed at concentrating the non-zero elements near a diagonal of the data array, to simplify proper clustering of data performed by SCM.

The α-clique method maximizes the quality criterion (2) and gives final clustering as a result of the specialized EA.

The obtained results for the symmetrized data are presented in Figs. 12 and 13; data clustering and its interpretation on the maps of Indonesia (Figs. 14 and 15), while results for the non-symmetric data are presented in Figs. 14 and 15.

Generally, both used methods give satisfying results, but the method based on α-cliques is much more flexible and the obtained clusters are homogeneous

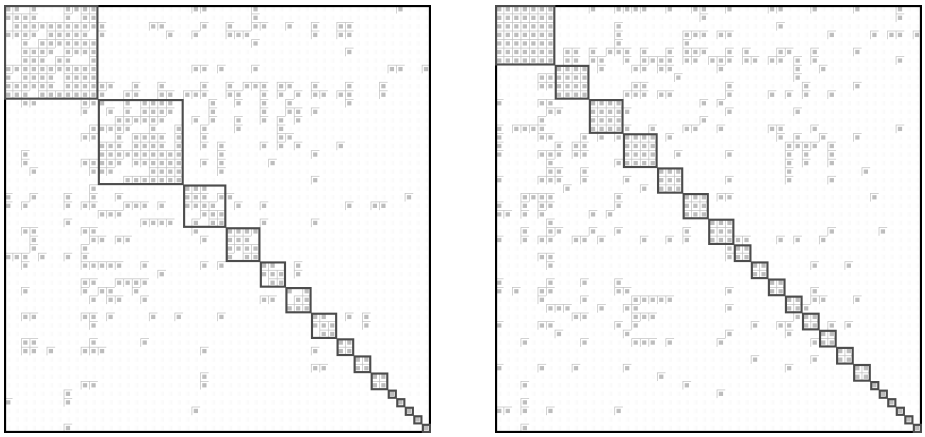


Figure 12. The import-export solutions obtained for symmetrized data using the α-clique method with $\alpha = 0.7$ (left) and $\alpha = 1$ (right)

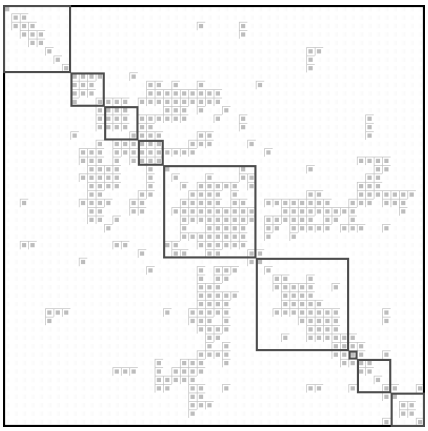


Figure 13. The import-export solution obtained with a DSM matrix method of clustering for symmetrized data using evolutionary algorithm (EA) and simple clustering method (SCM)

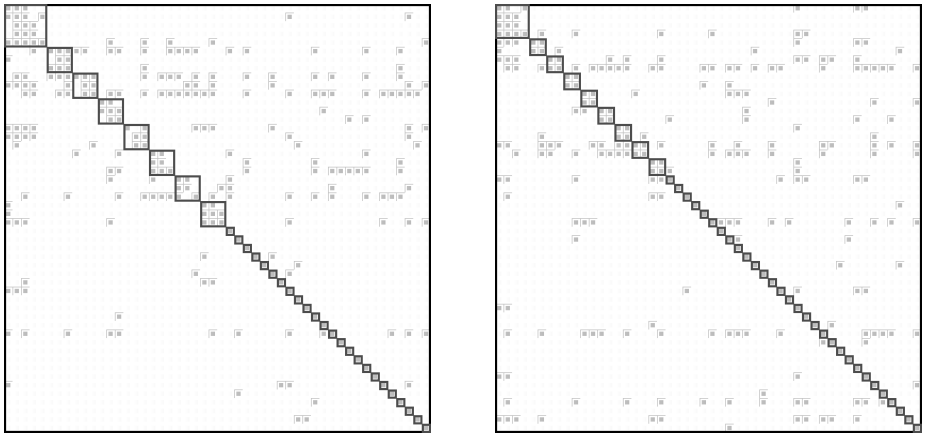


Figure 14. The import-export solution obtained for non-symmetric data using the α-*clique* method with $\alpha = 0.51$ (left) and $\alpha = 0.7$ (right)

and with more equal size, with a guaranteed level of inner connections. The DSM matrix method gives one result, with small possibilities to tune it - as a result of the EA a population of several solutions is obtained, thus it is possible to take into account other, rather worse solutions, but in the α-clique-based method we can do this as well, beside tuning the parameter α. The parameters of the detected clusters are rather unstable. As it can be seen in Fig. 15, there are very dense and very sparse clusters, thus the level of connections among their

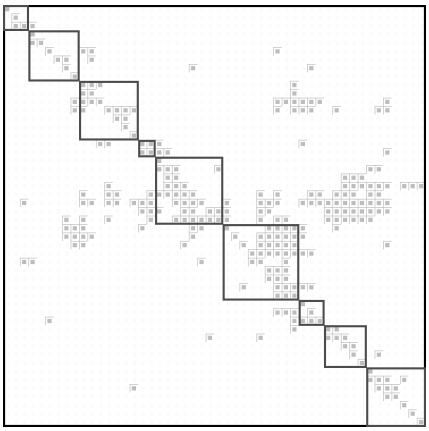


Figure 15. The import-export solution obtained with a DSM matrix method of clustering for non-symmetric data using evolutionary algorithm (EA) and simple clustering method (SCM)

elements is not guaranteed. Also the DSM matrix method is more complicated (computations last longer for the same problem), contains two stages, while the α-clique-based method gives the final results. Of course, the α-clique method requires tuning the α parameter to obtain satisfying results.

4.2.2. The problem of airport design

Here, nodes are essential airport units, edges among them are flows of passengers, cargo, luggage etc. Figs. 18 and 19 show the results obtained using the initial data presented in Fig. 11. Fig. 19 presents the results obtained using EA as a preprocessing tool and extracting clusters using SCM. SCM detects deep minimums in the histogram of data from preprocessed matrix and according to this, generates clusters. Fig. 18 presents results generated by the α-clique based method. The number of extracted clusters is equal to that obtained by the SCM method, but computations are easier and faster. The complete clustering is a result of execution of one evolutionary method. Additional benefit of this method is that by changing parameters of the fitness function (for instance using (2) or (3) or similar) or α we can change the number of clusters and their sizes.

4.2.3. An example of the influence of parameter α on α-clique graph covering

As the first step of the α-clique application to graph covering, we tested the influence of α value on parameters of obtained clusters. The results for two quite

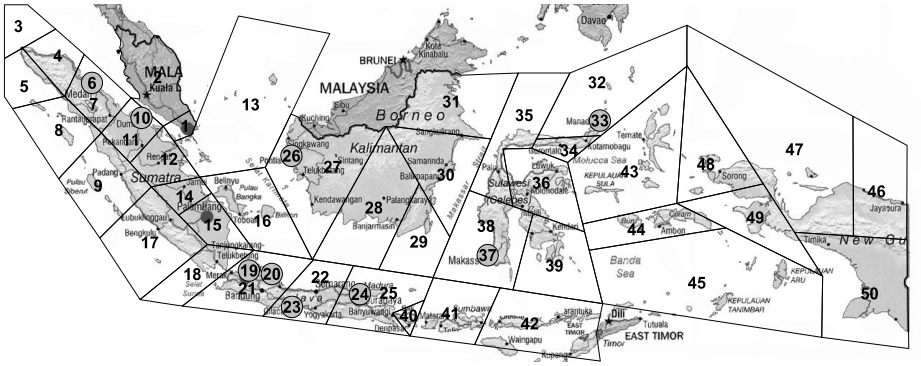


Figure 16. The map of Indonesian regions

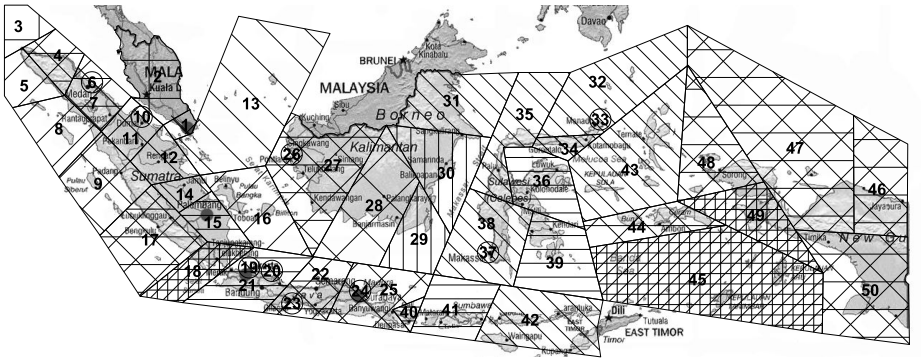


Figure 17. The map of trade connections among Indonesian regions

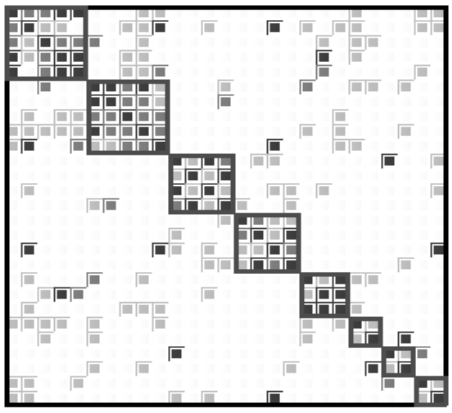


Figure 18. The airport design solution obtained using the α-clique method with $\alpha = 0.8$

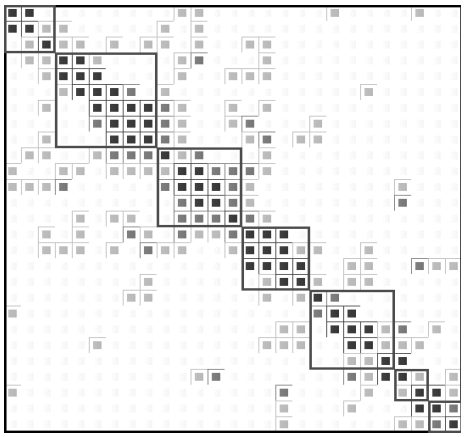


Figure 19. The airport design solution obtained with a DSM matrix method of clustering using evolutionary algorithm (EA) and the simple clustering method (SCM)

large graphs of 4 000 vertices and different numbers of edges (7 425 226 and 572 774) are presented in Tables 1 and 2. To give the possibility of comparing the obtained data with more conventional method of finding α-cliques, we also present similar results calculated by a greedy algorithm. The applied greedy algorithm (Algorithm 4) is a very simple method, starting from the randomly selected vertex. The first vertex constitutes the first α-clique. The subsequent vertices are also selected randomly and the algorithm tries to attach them to the previously selected α-cliques. If this fails, it creates a new α-clique. This procedure is repeated until all vertices are attached to α-cliques.

The results obtained using the greedy method are also presented in Tables 1 and 2. The greedy algorithm is rather fast, thus to obtain better solutions we have chosen the best result out of 50 simulations.

As it can be noticed, the first example graph (Table 1) is rather large, dense and almost complete, thus only values higher than α=0.8 are practically applicable. One simulation of the EA method with 10 000 epochs (iterations) and α=0.95 lasts abut 15.5 hours on a computer with processor AMD Athlon 64 4800+ (2500MHz) under LINUX operating system, but results remain almost unchanged during last 5000 iterations. The greedy method is faster, one simulation lasts about 5 minutes, but obtained results are significantly worse.

Table 1. Results obtained for a test graph with 4000 vertices and 7 425 226 edges.

α	Min α-clique	Max α-clique	Number of α-cliques	Average size of α-clique	Min α-clique	Max α-clique	Number of α-cliques	Average size of α-clique
	EA method				Greedy method			
0.80	4000	4000	1	4000	2	1572	18	66
0.90	41	607	15	267	2	504	31	47
0.95	1	277	39	103	2	156	51	73
0.97	8	131	56	71	2	106	67	63
0.99	5	74	78	51	2	70	83	56
1.00	10	73	76	53	2	70	84	54

Table 2. Results obtained for a test graph with 4000 vertices and 572 774 edges.

α	Min α-clique	Max α-clique	Number of α-cliques	Average size of α-clique	Min α-clique	Max α-clique	Number of α-cliques	Average size of α-clique
	EA method				Greedy method			
0.51	40	119	50	80	1	70	352	11
0.60	6	111	67	60	1	70	428	8
0.70	2	80	94	43	1	41	411	8
0.80	40	40	100	40	1	40	440	8
0.90	40	40	100	40	1	40	470	7
0.95	40	40	100	40	1	40	470	7

The second considered graph is also rather large, but very sparse and only application of values below 0.8 has an influence on the results. Generally, the number of obtained α-cliques increases and their size decreases in α. Thus, by modifying the value of α it is possible to change the sizes of the obtained separate α-cliques used for graph covering and tune the parameters of clustering to actual needs. One simulation for the second graph, performed with identical conditions as for the first one, lasts about 12 hours, but after less than 1000 epochs (or less than one hour) probably optimal graph clustering is obtained with 100 identical 40-element clusters. As it can be noticed, the problem complexity is connected mainly with the number of edges, the number of graph vertices has lower influence on computation time. The greedy method in the case of sparse graph gives even worse results than in the case of the dense one. Memory requirements for the data processed by the algorithm are similar for both examples (data are stored in the adjacency matrix; its size depends only

on the number of vertices) and can be expressed as:

$$M = (l^2 + m * l) * 4 \ [B] \tag{10}$$

where:
l – the number of all vertices in the considered graph (4000);
m – the number of all individuals in the EA (500);
which gives about 68.7 MB (data stored as 4-byte numbers).

Algorithm 4 - The greedy algorithm

Input:
$G(V, E)$ input graph
Output:
Q set of $Q_i(V_i, E_i)$ - α-cliques of G such that $V = \bigcup_i V_i$ and $V_i \cap V_j = \oslash$ for $i \neq j$

```
Q = ∅;
Q_i(V_i, E_i) : V_i = ∅ and E_i = ∅;
Vp = V
i = 1
while (Vp ≠ ∅)
    begin
        V_i = ∅;
        Ei = ∅;
        pick v_j ∈ V_p;
        if (χ(G_i, v) constitute α-cliques)
            begin
                χ(G_i, v);
                κ(G, v);
            end
        else
            begin
            Q = Q ∪ Q
            i := i + 1
        end;
    end;
```

$G_1(V_1, E_1)$ is a subgraph of $G(V, E)$
$\chi(G_1, v)$ is a function adding vertex $v \in V \backslash V_1$ to V_1 and relevant edges existing in the new G_1
$\kappa(G, v)$ is a function removing vertex v from V and every edge $e \in E$ such as $v \in e$.

5. Conclusions

In this paper we proposed two methods for solving problems connected with clustering in graphs, where the main aim is to detect groups of densely connected vertices. We also presented results of computational experiments that illustrate the properties of the proposed methods.

The first presented method is an extension of a method proposed by Lenstra (Lenstra, 1977). We added an evolutionary-based method to find the desired structure of the data matrix and then a simple algorithm to divide it into clusters.

The second presented method is based on our new concept of α-clique. The concept of α-clique gives new possibilities of separating hardwired structures from considered data, but determining α-cliques is a problem of large-scale of complexity. Thus, it seems justified to apply evolutionary algorithm to solve it. Experimental results confirm that applying α-cliques to detect concentrations of connections among objects yields acceptable solutions and using a specialized evolutionary algorithm makes it possible to obtain solution in reasonable time.

The results of the series of conducted experiments are rather encouraging, the parameter α introduced to modify the traditional notion of a clique gives a flexible tool that enables solving of the graph clustering problem.

References

Aho, A.V., Hopcroft, J.E. and Ullman, J.D. (1974) *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

Altus, S.S., Kroo, I.M. and Gage, P.J. (1996) A Genetic Algorithm for Scheduling and Decomposition of Multidisciplinary Design Problems. *J. of Mechanical Design*, **118**(4), 486–489.

Aussiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A. and Protasi, M. (1999) *Complexity and Approximation*. Springer.

Bagirov, A.M. and Yearwood, J. (2006) A new non-smooth optimization algorithm for minimum sum-of-squares clustering problems. *EJOR* **170**, 578–595.

Benson, S.J. and Ye, Y. (2000) Approximating maximum stable set and minimum graph coloring problems with the positive semidefinite relaxation. In: M. Ferris and J. Pang, eds., *Applications and Algorithms of complementarity*. Kluwer Academic Publishers, 1–18.

Beineke, L. and Wilson, R. (1978) *Selected Topics in Graph Theory*. Academic Press.

Browning, T.R. (2001) Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions. *IEEE Transactions on Engineering Management*, **48**(3), 292–306.

Chen, Z.-Q., Wang, R.-L. and Okazaki, K.(2008) An Efficient Genetic Algorithm Based Approach for the Minimum Graph Bisection Problem. *IJCSNS International Journal of Computer Science and Network Security*, **8**(6), 118–124.

Cichosz, P. (2000) *Systemy uczce się* (Learning systems, in Polish). WNT, Warszawa.

Cowen, L. (1998) *Approximation Algorithms.* John Hopkins University.

FALKNER, J., RENDL, F. and WOLKOWICZ, H. (1994) A computational study of graph partitioning. *Mathematical Programming*, **66** (2), 211–239.

HANSEN, P., MLADENOVIC, N. and UROSEVIC, D. (2004) Variable neighborhood search for the maximum clique. *The Fourth International Colloquium on Graphs and Optimisation* (GO-IV), **145** (1), 117–125.

HASSIN, R. and KHULLER, S. (1986) z-Approximation. *Journal of Algorithms*, **41** (2), 429–442.

HOCHBAUM, D.S., ed. (1997) *Approximation Algorithms for NP-hard Problems* PWS Publishing Company.

HROMKOVIC, J. (2001) *Algorithmics for Hard Problems*. Springer.

JAIN, A.K. and DUBES, R.C. (1988) *Algorithms for Clustering Data*. Eaglewood Cliffs, NJ, Prentice-Hall.

JUKNA, S. (2001) *Extremal Combinatorics*. Springer-Verlag, Berlin-Heidelberg.

KORTE, B. and VYGEN, J. (2000) *Combinatorial Optimization, Theory and Algorithms*. Springer.

KUMLANDER, D. (2007) An Approach for the Maximum Clique Finding Problem, *Test Tool Software Engineering*. Software Engineering, Innsbruck, Austria.

LENSTRA, J.K. (1977) Sequencing by enumerative methods. *Mathematical Centre Tracts*, Amsterdam.

MARCHIORI, E. (1998) A Simple Heuristic Based Genetic Algorithm for the Maximum Clique Problem. *Proceedings of the 1998 ACM Symposium on Applied Computing*. ACM, 366–373.

MCCULLEY, C. and BLOEBAUM, C.A. (1996) Genetic Tool for Optimal Design Sequencing in Complex Engineering Systems. *Structural Optimization*, **12** (2-3), 186–201.

MCCORMICK, W.T., SCHWEITZER, P.J. and WHITE, T.W. (1972) Problem decomposition and data reorganization by a clustering technique. *Operations Res.* **20**, 993–1009.

MICHALEWICZ, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin-Heidelberg

MULAWKA, J. and STAŃCZAK, J. (1999) Genetic Algorithms with Adaptive Probabilities of Operators Selection. *Proceedings of ICCIMA'99*, New Delhi, India. World Scientific, 464–468.

POTRZEBOWSKI, H., STAŃCZAK, J. and SĘP, K. (2004) Evolutionary method in grouping of units with argument reduction. *Proceedings of the 15th International Conference on Systems Science vol. 3*. Oficyna Wydawnicza Politechniki Wrocławskiej, 29-3-6.

POTRZEBOWSKI, H., STAŃCZAK, J. and SĘP, K. (2006) Evolutionary Algorithm to Find Graph Covering Subsets Using α-Cliques, In: J. Arabas, ed., *Evolutionary Computation and Global Optimization*, Prace Naukowe Politechniki Warszawskiej, 351–358.

PROTASI, M. (2001) Reactive local search for the maximum clique problem. *Algorithmica*, **29**(4), 610–637.

Rogers, J.L. (1997) Reducing Design Cycle Time and Cost Through Process Resequencing. *International Conference on Engineering Design ICED 1997*, Tampere, Finland, Estonian Academy Publishers.

Stańczak, J. (2003) Biologically inspired methods for control of evolutionary algorithms, *Control and Cybernetics*. **32**(2), 411–433.

Sysło, M.M., Deo, N. and Kowalik, J.S. (1983) *Algorithms of Discrete Optimization*. Prentice-Hall.

Talbi, E.-G. and Bessiere, P. (1991) A parallel genetic algorithm for the graph partitioning problem. *Proc. of the 5th International conference on Supercomputing*. ACM, New York, 312–320.

Williamson, D. (1999) Lecture Notes on Approximation Algorithms. *IBM Research Report RC* 21409 02 17.

Wilson, R.J. (1996) Introduction to Graph Theory. Addison Wesley Longman.

Yu, T.L., Goldberg, D.E., Yassine, A. and Yassine, C. (2003) A Genetic Algorithm Design Inspired by Organizational Theory. *Genetic and Evolutionary Computation Conference (GECCO) 2003, Chicago, Illinois*. Springer-Verlag, Heidelberg, **LNCS 2724**, 1620–1621.