Proc. Estonian Acad. Sci. Engin., 1995, 1, 2, 113–138
https://doi.org/10.3176/eng.1995.2.02

EXTRACTING OF ALL MAXIMAL CLIQUES: MONOTONE SYSTEM APPROACH

Rein KUUSIK

Tallinna Tehnikaülikooli Informaatikainstituut (Department of Informatics, Tallinn Technical University), Raja 15, EE-0026 Tallinn, Eesti (Estonia), e-mail: kuusik@cc.ttu.ee

Presented by L. Mõtus

Received 8 November 1995, accepted 31 January 1996

Abstract. NP-complicated problems have been described in the graph theory. An example is the extracting of all maximal cliques from a graph. Many algorithms for solving this problem have been described. However, complexity is linear to the number of maximal cliques. This paper discusses a new approach for extracting all maximal cliques, based on the monotone system theory. The complexity of the presented algorithms is linear to the number of maximal cliques.

Key words: theory of monotone system, graph, clique.

1. INTRODUCTION

Assume that the finite undirected simple graph $G(V, E)$ is given, where V is the set of nodes, $|V| = N$, E is the set of edges.

Definition G1. *The arbitrary full graph is called a clique.*

Definition G2. *The clique which does not contain other cliques is called a maximal clique.*

Definition G3. *The largest maximal clique is called a maximum clique.*

Problem. To extract all maximal cliques from the graph G.

Many algorithms have been described to solve this problem. The best solution today is an algorithm, the complexity of which is linear to the number of maximal cliques [1, 2]. The theory and algorithms described in this paper can solve this problem. We assume that the graph G is presented in the form of an adjacency matrix $X(N, N)$, the main diagonal of which has zeros.

2. THE THEORY OF MONOTONE SYSTEM

This section gives the main concepts of the theory of monotone system [3].

Definition M1. *Let a finite discrete set* X, $|X| = N$ *and a function* π_X *on it, which maps to each element* $b \in X$ *a certain nonnegative number (weight)* $\pi_X(b)$, *be given. The function* π_X *is called a weight function if it is defined on any subset* $X' \subseteq X$; *the number* $\pi_{X'}(b)$ *is called a weight of element* b *in* X'.

Definition M2. *A set* X *with a weight function* π_X *is called a system (or a system on elements from* X*) and is denoted by* $\Pi = (X, \pi_X)$.

Definition M3. *The system* $\Pi' = (X', \pi_{X'})$, where $X' \subseteq X$ *is called a subsystem of the system* $\Pi = (X, \pi_X)$.

Definition M4. *The system* $\Pi = (X, \pi_X)$ *is called monotone if in the case of any* $b \in X'\backslash\{c\}$, $c \in X$, $\pi_{X'\backslash\{c\}}(b) \leq \pi_{X'}(b)$, *where* X' *is any subset of* X.

Definition M5. *Function* Q *which maps to every subset* $X' \subseteq X$ *of the monotone system a nonnegative number* $Q(X') = \min \pi_{X'}(b)$ *is called an objective function.*

Definition M6. *The subsystem* $\Pi = (W, \pi_W)$ *of the monotone system* $\Pi = (X, \pi_X)$, *in the case of which the function* Q *obtains a maximal value* $Q(W) = \max Q(X') = \max \min \pi_{X'}(b)$, *is called a kernel of the monotone system* Π; *respective* $Q(W)$ *value is called a measure of the kernel quality.*

2.1. How to create a monotone system

To use the method of monotone system, we have to fulfil two conditions:

1) there has to be a weight function $\pi_X(b)$ which will give a measure of influence for every element b of the monotone system X;

2) there have to be rules F to recompute the weight of the elements of the system if there is a change in the weight of one element.

These conditions give us a lot of freedom to choose the weight functions, and the rules of weight change in the system. The only constraint we have to keep in mind is that the rules F and the weight function π have to be compatible in the sense that after eliminating all elements b from the system X, the final weights of $b \in X$ must be equal to zero.

3. MONS-ALGORITHM

In this section we describe a greedy algorithm for extracting all maximal cliques from an undirected finite graph. It uses all main steps of the monotone system algorithms. It is a MONS-Algorithm [4, 5] in the graph theory. In essence, A1 is a recursive algorithm. We present its back-tracking version here.

To separate all maximal cliques from a graph $G(V, E)$, we use its adjacency matrix X_i (index i shows the number of matrix generation, X_0 corresponds to G), on the main diagonal of which we use zeros. A monotone system will be created on X. In this algorithm as a weight of a node, we use its degree in the subgraph G_i, corresponding to the adjacency matrix X_i. It is easy to check that these weights create a monotone system on a graph. All main theorems (weight function is monotone, the maximal clique as a set of elements of the adjacency matrix is a kernel of the monotone system) were proved in [4].

The weights (degrees) of the nodes will be saved in the vector F_i. The nodes belonging to a clique in the generation i are saved in the vector CLIQUE_i.

In all generations we analyse only nodes with non-zero weights. The nodes in the list CLIQUE_i are not used for the matrix X_i analysis. Thus, in the last generation (the possible maximal number of generations equals to the number of nodes in the maximum clique) the number of nodes to be analysed equals to one. The process of elimination of a node in the generation i means that this node is eliminated from the analysis also for all next generations created from this generation i. The maximal clique is found if

1) the weight of a node in X_i has been changed to zero or

2) all non-zero weights in F_i are equal to the number of nodes under analysis in X_i minus one (N nodes with a weight N–1 each one, the so called "rule n–1").

3.1. Algorithm A1

P1. Initialising. i: = 0, CLIQUE_0: = { }, F_0: = { }.

P2. Calculation of weights of the nodes of X_0 to F_0.

P3. Check of a maximal clique. IF the weight of the node K in F_i is zero, we have found a maximal clique. Output CLIQUE_i and the node K. We shall output as many maximal cliques as many nodes K with zero-weight there exist in X_i.

P3A. IF the weights of the nodes in X_i are equal to the number of nodes with non-zero weight –1 THEN BEGIN a maximal clique has been found. Output all nodes with non-zero weight of X_i and CLIQUE_i as a maximal clique. Set all weights in F_i to zero END

P4. Control of back-tracking. IF all weights in F_i are zeros THEN

P4A. BEGIN i: = i – 1. IF i = –1 THEN GOTO FINISH ELSE GOTO P3A END

P5. Selection of the leading node. $CLIQUE_{i+1}$: = $CLIQUE_i$. Find the node J with the maximal weight in F_i. If there are more than one, take the first. Add J to the $CLIQUE_{i+1}$.

P6. Elimination of node and recalculation of weights. Eliminate the node J and recalculate weights in F_i (set zero to the weight of node J in F_i and diminish its neighbours' weights by one).

P7. Formation of a new generation. i: = i + 1. Exclude the submatrix X_i corresponding to the node J from X_{i-1}. Calculate the weights for F_i.

P8. Control of the exhaustedness of nodes. Compare weights of F_i and F_{i-1}. For nodes with equal weights in F_i and F_{i-1}, set their weights in F_{i-1} to zero and then diminish their neighbours' weights in F_{i-1} by one.

P9. Control of the originality of the extracted clique. IF there exists a node which has belonged to any extracted maximal clique, but does not belong to $CLIQUE_i$ and X_i, and which is adjacent to all nodes in $CLIQUE_i$ and X_i THEN GOTO P4A ELSE GOTO P3.

FINISH

Commentary to step P9. Although node elimination by diminishing its neighbours' weights by one, for a certain node, does not exclude its non-zero weight after extraction of all its maximal cliques. In this situation, extraction of a clique as a part of extracted maximal clique is possible. To exclude that, special testing (step P9) is needed. Theorem 5 below explains this in detail.

3.2. Proof of the correctness of algorithm A1

We present here the definitions and theorems proving the correctness of algorithm A1. The theorems are commented where necessary. We assume that the graph $G(V, E)$ is finite and undirected. In the description of algorithm A1 we preferred maximal weight in choosing the leading node. In the theorems proved below, we assume that the node may be chosen to be a leading node independent of its weight.

3.2.1. Definitions

Definition 1. *The adjacency matrix of the node Y is called an extract by Y. The node Y is called a leading node.*

Definition 2. *Suppose that the graph G and its adjacency matrix X(N, N) are given. Assume that we have made an extract by arbitrary node* Y_0. *This extract describes a subgraph* $G_1 \subset G$. *Denote its adjacency matrix by* X_1, $X_1 \subset X$, $V_1 \subset V$, $|V_1| = N_1$. *Now we can make an extract from* X_1. *Denote it by* X_2, $X_2 \subset X_1 \subset X$ *and subgraph by* G_2, $G_2 \subset G_1 \subset G$,

correspondingly. In that way, we form a sequence of extracts $X \supset X_1 \supset X_2 \supset ... \supset X_t \supset ... \supset X_f$, $f \leq N-1$, $|V_f| = N_f$. *That sequence determines the fixed sequence of the leading nodes* $Y_0, Y_1, Y_2, ..., Y_{f-1}$. *The adjacency matrix* X_t *is called an extract on level* t, $0 < t < N$, *and the leading node, by which we make extract* $X_{t+1} \subset X_t$, *is called a leading node on level* t.

We call the adjacency matrix X of the graph G an extract on level 0 and denote it by X_0.

Definition 3. *The node of the graph G, which has not been chosen to be a leading node on level t but all its cliques in* X_t, *will be extracted until the back-tracking to level t is called the exhaustive node on level t.*

The adjacency matrix X_0 is getting smaller step by step because of the elimination of the leading nodes and exhaustive nodes on level 0.

3.2.2. Theorems

Theorem 1. *If for a certain node Z its weight (degree)* $F_{t+1}(Z) + 1 = F_t(Z)$, *then Z is an exhaustive node.*

Proof. Suppose that Y is a leading node by which the extract $X_{t+1} \subset X_t$ is made. If in X_{t+1} for a node Z $F_{t+1}(Z) + 1 = F_t(Z)$ ($F_{t+1} + 1$ because of zeros on the main diagonal), it means that all node Z neighbours in X_t have Y neighbours too. If all maximal cliques for the node Y are separated, then all maximal cliques for Z are also separated. Thus, according to Definition 3, the node Z is an exhaustive node.

The theorem is proved.

Commentary. This theorem is deeper in essence than it may seem. In the extracting process, the matrix X_t becomes smaller and smaller, with the number of nodes decreasing. Thus, exhaustedness of X_t does not follow from forming of one sequence of the leading nodes $Y_t, Y_{t+1}, ...$. If the node Z has not been chosen as a leading node on level t, it will be exhausted completely relative to the leading node in X_t if $F_{t+1}(Z) + 1 = F_t(Z)$.

Theorem 2. *In algorithm* A1, *to every extract* X_t *corresponds a clique* $\{Y_0, Y_1, ..., Y_{t-1}, U\}$ *with R nodes. The clique is maximal, if*

1) *the degree (weight) of the node U in* X_t *equals to zero,* $R = t + 1$ *or*

2) *in* X_t *the degree of nodes* $\{j\}$ *with non-zero weight (their number* $A \leq N_t$*) equals to* $A - 1$ *(so called "rule n – 1"). In the latter case* $U = \{j\}$, $R = t + |U|$.

Proof. By algorithm A1, the leading node Y_{t-1} is chosen from nodes of X_{t-1}, $t > 0$, and the extract X_t, $X_t \subset X_{t-1} \subset ... \subset X_0$ is made. It guarantees that the nodes of the sequence $Y_0, Y_1, ..., Y_{t-1}$ are neighbours. Thus on level $t - 1$, we have extracted a clique $\{Y_0, Y_1, ..., Y_{t-1}\}$ with t nodes.

The leading node chosen on level $t - 1$ is eliminated from X_{t-1}. Back-tracking to the level $t - 1$ takes place only when X_t is empty. It means

that every node of X_t if it is not an exhaustive node will be a leading node on level t.

Next, we prove the maximality of the extracted cliques. We assume that for every X_t, step P9 of algorithm A1 is applied.

1) If the leading node Y_{t-1} from X_{t-1} has been chosen, it is eliminated from X_{t-1}. If then the weight of any node $U \neq Y_{t-1}$ in X_{t-1} changed to zero (consequently also in X_t), then the nodes $Y_0, Y_1, \ldots, Y_{t-1}, U$ are neighbours and they have no neighbours together in G. Consequently, the clique with $t+1$ nodes is maximal.

2) If we have a situation, where all nodes $\{j\}$ of X_t with non-zero weights have a weight equal to $A-1$ (because the main diagonal of X_t has zeros) it means that those nodes are neighbours. As $X_t \subset X_{t-1} \subset X_{t-2} \subset \ldots \subset X_1 \subset X_0$, then members of the sequence $Y_{t-1}, Y_{t-2}, \ldots, Y_0$ are neighbours of these A nodes and there are no common neighbours of them in X_t. Thus, the clique from these $t+A$ nodes is maximal.

The theorem is proved.

Theorem 3. *Suppose that we are on level t and the leading node Y_t has been chosen. Then, relative to Y_t, all maximal cliques included in X_t are extracted.*

Proof. A1 is a recursive algorithm in which all activities are the same for any extract X_t. The extract X_t can not be exhausted before all extracts $X_{t+1} \subset X_t$ would exhaust.

On the basis of Y_t, the extract $X_{t+1} \subset X_t$ is made. It includes only nodes adjacent to Y_t in X_t. The back-tracking to X_t would happen only if all nodes of X_{t+1} were chosen as the leading ones on X_t or they were exhaustive nodes (see Def. 3), i.e. when $X_{t+1} = \varnothing$. This means that we have obtained all pairs of nodes $Y_t \rightarrow Z_{t+1}$, where Z_{t+1} is an arbitrary node of X_{t+1}. Here any node of X_{t+1} can be chosen or exhaustive only at once in X_{t+1}.

It happened so on any level i, $t < i \leq N$. This technique guarantees that relative to Y_t, we analyse all possible combinations of its neighbours in X_t. Since $X_t \supset X_{t+1} \supset X_{t+2} \supset \ldots$, then we analyse only combinations between adjacent nodes, i.e. $Y_t \rightarrow Y_{t+1}$, $Y_t \rightarrow (Y_{t+1} \rightarrow Y_{t+2})$, $Y_t \rightarrow (Y_{t+1} \rightarrow (Y_{t+2} \rightarrow Y_{t+3}))$, etc. The current sequence of extracts $X_t \supset X_{t+1} \supset X_{t+2} \supset \ldots$ ends if the elements of the current leading node sequence have no common nodes, i.e. $X_{t+i} = \varnothing$. That is exactly when the maximal clique has been extracted (see Theorem 2). Then the back-tracking to X_{t+i-1} is made and all the described activities are repeated again but relative to X_{t+i-1}.

Thus, using algorithm A1 relative to Y_t, all maximal cliques included in X_t are separated.

The theorem is proved.

Corollary 3.1. If the leading node Y_t is chosen, we may nullify its weight in X_t before all maximal cliques relative to Y_t in X_t are extracted.

Theorem 4. *Algorithm* A1 *excludes repetitive maximal clique extracting.*

Proof. Suppose that we are on level t and we have chosen a leading node Y_t for extracting X_{t+1}. Y_t is eliminated from X_t. Consequently, in back-tracking to level t the node Y_t cannot be included in the next $X_{t+1} \subset X_t$. It happens so on every level t, $t \geq 0$. Thus, for any level t in extracting of the maximal clique relative to a leading nodes sequence $Y_0, Y_1, \ldots, Y_t$, any two cliques from X_t cannot be identical.

The theorem is proved.

3.3. An example of using algorithm A1

Suppose that the graph G is given by adjacency matrix X:

X	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	1	1	1	0
3	1	1	0	1	0	1
4	0	1	1	0	1	1
5	0	1	0	1	0	0
6	0	0	1	1	0	0

On extracting all maximal cliques, we use algorithm A1. To demonstrate the work of the algorithm, we choose the graph on which the originality check of the extracted clique (algorithm A1 step P9) is not needed.

$\text{CLIQUE}_0 = \{\}$. According to step P2, we calculate its degree (weight) to every node in X. On this step, we form the vector of weights F_0:

F_0: 2 4 4 4 2 2

Then we check if the "rule $n-1$" (P3A) applies.

Since X is not a clique, the node with the greatest degree in X is chosen to the leading Y_0 (P5). As there are more than one, we choose the first, it means $Y_0 = 2$: $\text{CLIQUE}_1 = \{2\}$. The leading node is eliminated (P6), it means its weight in F_0 is zero-filled and its neighbours' weights in X are decreased by one:

F_0: 1 0 3 3 1 2

According to step P7, the extraction X_1 is made by $Y_0 = 2$ and the weights F_1 are calculated. Then the feedback comparison of weights is made (P8).

X_1	1	3	4	5
1	0	1	0	0
3	1	0	1	0
4	0	1	0	1
5	0	0	1	0
F_1:	1	2	2	1

The feedback comparison is applied and the nodes 1 and 5 are eliminated from level 0. The recalculated weights on X_0 are:

F_0: 0 0 2 2 0 2

Now according to the step P3, we have to check the existing of zero-weights on X_1. As all nodes have non-zero weights, we must choose the next leading node Y_1 (P5). $Y_1 = 3$, CLIQUE$_2$ = {2,3}. We eliminate it (P6):

F_1: 0 0 1 1

Then we extract X_2 and calculate its node degrees (P7):

X_2	1	4
1	0	0
4	0	0
F_2:	0	0

As we can see, the nodes of X_2 have a weight equal to zero (P3). Thus, we have separated two maximal cliques

{2, 3, 1} and {2, 3, 4}.

As there are no more nodes with non-zero weight in X_2, we have to do back-tracking (P4) to level 1. But the nodes with zero-weights will not be analysed here.

X_1	4	5
4	0	1
5	1	0
F_1:	1	1

The "rule $n-1$" is applied (P3A). It means that the maximal clique {2, 4, 5} has been separated. Since all nodes on level 1 now have weight = 0, we do back-tracking to level 0 (P4). Three nodes are analysed:

X_0	3	4	6
3	0	1	1
4	1	0	1
6	1	1	0
F_0:	2	2	2

The "rule $n-1$" is applied (P3A), which means separating the maximal clique $\{3, 4, 6\}$ and zero-filling the weights of those nodes. As this is level 0 and there are no nodes with non-zero weight, then all maximal cliques from G have been separated.

3.4. Why do we need step P9 of algorithm A1

The last theorem proved that every extracted maximal clique by algorithm A1 is unique. Why is the control of clique originality needed then? The reason is that the clique extracted by A1 may be a clique, but not a maximal clique. This situation is possible because of decreasing the neighbours' weights by one when eliminating the leading node Y_t does not exclude the change of neighbour Z of Y_t into an exhaustive node but its weight $F_{t+1}(Z) + 1 \neq F_t(Z)$.

The next theorem will explain the described situation. Let us define some new terms.

Definition 4. *Suppose that X_{t+1} has been extracted. Any node Z which has belonged to any extracted maximal clique and does not belong to the leading node sequence $Y_0, Y_1,..., Y_t$ is called a banned node. All the other nodes are called feasible nodes.*

Theorem 5. *Suppose that the leading node Y_t from X_t was chosen, and $X_{t+1} \subset X_t$ was extracted. If there exists a banned node j, relative to which all maximal cliques were extracted in graph G and which was connected with all leading nodes in the current sequence $Y_t, Y_{t-1},..., Y_0$ and with all nodes of X_{t+1} , then the clique to be extracted is a part of the maximal clique extracted earlier.*

Proof. Suppose that there exists a banned node j which was chosen to be a leading node on level 0 and which was connected with all leading nodes $Y_t, Y_{t-1},..., Y_0$ and with all nodes of X_{t+1} . For the banned node j, all maximal cliques were extracted. If it is connected with all $\{Y_i\}$, $i = 0,..., t$, and with all nodes $\{p\}$ of X_{t+1}, then it means that all these nodes $\{Y_i\}$ and $\{p\}$ and j together were analysed earlier in extracting maximal cliques of j. Consequently, nodes $\{Y_i\}$ and $\{p\}$ together can form a clique from the maximal clique extracted earlier.

The theorem is proved.

To exclude the situation described in Theorem 5, a special test is needed (step P9 in A1). It is applied to every X_t and therefore it is time-consuming. If the result of that activity is positive, then the sequence of extracts X_{t+1} is interrupted and the back-tracking to X_t is made.

4. WHEN DO WE USE THE CLIQUE ORIGINALITY TESTING

Next, we will show that this kind of testing is needed only in certain cases.

Definition 5. *A node which has not been compared with the node Z is called a feasible node relative to Z.*

Applying this definition in extracting all maximal cliques, we have to add new activities to algorithm A1.

First, we have to choose a leading node among the feasible ones. It guarantees that the maximal clique, which includes a feasible node, is original and does not need special testing of originality.

We form the thesis as a theorem.

Theorem 6. *The sequence of the leading nodes which includes a feasible node corresponds to a maximal clique not yet extracted.*

Proof. According to Definition 4, the feasible node is a node which has not belonged to any extracted maximal clique. Thus the maximal clique, which includes a feasible node, is original. If the sequence of leading nodes includes a feasible node related to *Y*, then assuming that the node *Y* is in this sequence, it is the sequence of nodes which has not been analysed earlier. Thus, the extracted maximal clique is original. The maximality of the extracted clique is guaranteed by Theorem 2.

The theorem is proved.

Because of the conflict between a node's weight and the elimination process, a situation may occur where for a node *Z*, all maximal cliques are extracted (we do not know that!), but the weight of *Z* is not equal to zero. The situation is described by the next theorem.

Theorem 7. *If in* X_0 *there exists a node Y which has no feasible neighbours, but*

1) *the weight of Y is not equal to zero and*

2) *there do not exist banned nodes which are adjacent to Y and to all the nodes of an extract based on it,*

then there exists unextracted maximal clique(s) in graph G.

Proof. For nodes, according to Theorem 3, with weights equal to zero on level 0, all maximal cliques have been extracted. The node which has feasible neighbours has unextracted maximal cliques (Theorem 6). For other nodes with non-zero weight on level 0, there could exist unextracted maximal cliques. According to Theorem 5, it can only be a node which has no common banned node with nodes of extract based on it.

The theorem is proved.

To apply results of Theorem 7, we have to choose a feasible node to be a leading one. If no nodes of this kind exist, we have to choose a node with feasible neighbours.

According to Theorem 7, we have to use the test of clique originality (step P9) only if there exist some nodes with non-zero weights and without feasible neighbours on level 0. In this case, we can choose any node as a leading one and algorithm A1 step P9 guarantees that the maximal clique extracted is unique.

We use the results of Theorem 7 in algorithm A2.

4.1. Algorithm A2

P1. Initialising. $i := 0$, $\text{CLIQUE}_0 := \{\}$, $F_0 := \{\}$.

P2. Calculation of weights of nodes of X_0 to F_0.

P3. Check of a maximal clique. IF the weight of some node K in F_i is zero, we have found a maximal clique. Output CLIQUE_i and the node K. We shall output as many maximal cliques as many nodes K there exist in X_i.

P3A. IF the weights of the nodes in X_i are equal to the number of nodes with non-zero weight – 1 THEN BEGIN a maximal clique is found. Output all nodes with non-zero weight of X_i and CLIQUE_i as a maximal clique. Set all weights in F_i to zero END

P4. Banning of nodes and control for back-tracking. IF the maximal clique(s) was extracted THEN ban its nodes and ban these nodes relative to themselves. IF all weights in F_i are zeros THEN

P4A. BEGIN $i := i - 1$. IF $i = -1$ THEN GOTO FINISH ELSE GOTO P3A END

P5. Selection of the leading node. $\text{CLIQUE}_{i+1} := \text{CLIQUE}_i$. As a leading node, find a feasible node J with the maximal weight in F_i ($C = 1$). If there are no feasible nodes, find a node J with a maximal number of feasible neighbours ($C = 2$). If there are several nodes, take the node with the maximal weight. If there are no such nodes, take the first node with a maximal weight ($C = 3$). Add J to CLIQUE_{i+1}.

P6. Elimination of node and recalculation of weights. Eliminate the node J and recalculate weights in F_i (set zero to the weight of node J in F_i and diminish its neighbours' weights by one).

P7. Formation of a new generation. $i := i + 1$. Exclude the submatrix X_i corresponding to the node J from X_{i-1}. Calculate the weights for F_i.

P8. Control of the exhaustedness of nodes. Compare weights of F_i and F_{i-1}. For nodes with equal weights in F_i and F_{i-1}, set their weights in F_{i-1} to zero and then diminish their neighbours' weights in F_{i-1} by one.

P9. Control of the originality of the extracted clique. IF $C < 3$ THEN GOTO P3. IF there exists a node which belongs to any extracted maximal

clique, but does not belong to $CLIQUE_i$ and X_i, and which is adjacent to all nodes in $CLIQUE_i$ and X_i THEN GOTO P4A ELSE GOTO P3.

FINISH

4.2. An example of using algorithm A2

Suppose that we have a graph G with an adjacency matrix X:

X	1 2 3 4 5 6
1	0 1 1 0 1 0
2	1 0 1 0 0 1
3	1 1 0 1 0 0
4	0 0 1 0 0 0
5	1 0 0 0 0 0
6	0 1 0 0 0 0

To extract all maximal cliques, we use algorithm A2 here.

$CLIQUE_0 = \{\}$. At the beginning, all the nodes are feasible ones, i.e. FEASIBLE = {1, 2, 3, 4, 5, 6}, BANNED = {}.

According to step P2, we calculate the weights:

F_0: 3 3 3 1 1 1

Then we check the applicability of the "rule $n-1$" (P3A). As X is not a clique, the feasible node with the greatest weight would be chosen as the leading Y_0 (P5). Since there exist many nodes with the greatest weight, the first of them, in our case node 1, is chosen: $Y_0 = 1$, $CLIQUE_1 = \{1\}$. Then the leading node is eliminated, meaning that its weight is zero-filled and its neighbours' weight in X_0 is decreased by one (P6):

F_0: 0 2 2 1 0 1

According to step P7, the extraction X_1 is made and the weights are calculated, then feedback is compared (P8).

X_1	2 3 5
2	0 1 0
3	1 0 0
5	0 0 0

F_1: 1 1 0

The feedback comparison (P8) is not applicable. According to step P3, the existence of nodes with zero-weight in X_1 is checked. There is one node of this kind – node 5. Thus, the maximal clique {1, 5} has been extracted. Now, X_1 includes two nodes with non-zero weight. By applying the "rule $n-1$" (P3A), the maximal clique {1, 2, 3} is separated, and these nodes are banned:

BANNED = {1, 5, 2, 3}.

Since all nodes in X_1 have been analysed (all weights are zeroes), we do back-tracking to level 0 (P4A).

X_0	2	3	4	6
2	0	1	0	1
3	1	0	1	0
4	0	1	0	0
6	1	0	0	0
F_0:	2	2	1	1

As the "rule $n-1$" is not applied (P3A), node 4 is chosen as the first among feasible nodes with the equal greatest weights $\{4, 6\}$, $Y_0 = 4$, $\text{CLIQUE}_1 = \{4\}$ (P5) and eliminate it (P6):

F_0: 2 1 0 1

Then we extract X_1 (P7):

X_1	3
3	0
F_1:	0

As we see, there is only one node and it has the weight = 0, thus the maximal clique $\{4, 3\}$ has been extracted (P3), BANNED = $\{1, 5, 2, 3, 4\}$ (P4). Since X_1 has no more nodes to analyse, we do back-tracking to level 0 (P4A).

X_0	2	3	6
2	0	1	1
3	1	0	0
6	1	0	0
F_0:	2	1	1

Now we choose node 6 as a leading one, the only feasible node in X_0, $Y_0 = 6$, $\text{CLIQUE}_1 = \{6\}$ (P6) and extract X_1 (P7).

X_1	2
2	0
F_1:	0

As X_1 has a node with zero-weight, the maximal clique $\{6, 2\}$ is extracted (P3), BANNED = $\{1, 5, 2, 3, 4, 6\}$ (P4).

As X_1 has no more nodes to analyse, we do back-tracking to level 0 (P4A).

X_0	2 3
2	0 1
3	1 0
F_0:	1 1

Now there are no feasible nodes under analysis in X_0, neither are there any feasible nodes relative to members of X_0 (we compared nodes 2 and 3 together relative to the leading node 1). Thus, we have to check the originality of the clique to be extracted (P9). We have to check if a "cover node" exists on X_i, for which all maximal cliques have been extracted and which is adjacent to all nodes in the current leading nodes sequence $\{Y_0,...,Y_{i-1}\}$. If it exists, the association of nodes has been analysed together earlier and the clique to be formed cannot be maximal.

In our case, the leading nodes sequence is empty and on X_0 there exists a "cover node" 1 as a cover node relative to which all maximal cliques have been extracted:

X_0	2 3	1
2	0 1	1
3	1 0	1
F_0:	1 1	2

Consequently, the clique {2, 3} is not maximal and we have to do back-tracking. But as we are on level 0, it means that all maximal cliques in X have been separated.

5. THEOREM ABOUT CONVERGENCE OF ALGORITHMS A1 AND A2

Theorem 8. *Algorithms* A1 *and* A2 *extract all and only maximal cliques.*

Proof. A1 is a recursive algorithm for which all activities are the same for every extract X_t. By extracts, the hierarchical tree (HT) of clique nodes is formed. Nodes are added to the tree by one. The clique is maximal if a certain node of the HT is a leaf. In this case, the back-tracking to the previous node of HT is done, and a next maximal clique is extracted. We always obtain only a current branch of HT.

1. Convergence. The leading node Y_t and all exhaustive nodes are eliminated from X_t and their neighbours' weights are decreased by one in X_t. Thus, in back-tracking to level t, the number of nodes in X_t is decreasing.

The leading nodes of the current sequence $Y_0, Y_1,..., Y_t$ do not belong to X_{t+1} . The elimination of a leading node and exhaustive nodes from X_t decreases their neighbours' weights in X_t. It means that the weight of the node in X_t is less than in X_{t-1}. All nodes of X_t with zero-weight do not belong to X_{t+1}.

The back-tracking to level t is done when X_{t+1} is empty. It means that all its nodes have a weight equal to zero. Thus, the number of back-trackings to level t cannot be greater than the number of nodes with non-zero weight in X_t.

2. Maximality of cliques. According to Theorem 2, the extracted clique is maximal.

3. The originality of the maximal clique. According to Theorem 4, every maximal clique is extracted only once. Proceeding from Theorems 5 and 7, we can capture the situations when the unoriginal clique is extracted. In that case, the special control is done.

The theorem is proved.

6. COMPLEXITY OF ALGORITHMS A1 AND A2

Before we present a theorem about the complexity of algorithms A1 and A2, we describe the main concept of these algorithms.

Algorithms form a n-tree with nodes, which are the nodes of maximal cliques. In reality, we do not form this tree but only pass it, we obtain only the branch of the current maximal clique. From any node of this tree, there may exit only so many branches as many maximal cliques are included in X_{t+1}.

The number of leaves of the n-tree was determined by the number of maximal cliques in the graph. The length of the longest branch of the n-tree equals to the number of nodes in the maximum clique of the graph G minus one. The length of a certain branch in the n-tree must not be equal to the number of nodes in the corresponding maximal clique because of the "rule $n-1$". In this case, the set of nodes corresponds to a leaf of the branch.

Thus, the efficiency of an algorithm depends on maximising the use of the "rule $n-1$" because of decreased need for the number of extracts X_t. In essence, it is a new optimising task to decrease the branch lengths (the number of extracts). In this case, the best algorithm will have a minimal number of nodes in the n-tree.

Algorithms A1 and A2 do not solve this optimisation task. They minimise the number of branches in the n-tree.

As one possibility, we can form its own n-tree for every maximal clique. But different maximal cliques may have common nodes. For algorithms A1 and A2, the goal is to maximise the number of common nodes for different cliques. The "rule $n-1$" is used here as a complementary instrument to diminish the number of nodes in n-tree.

Comparing these two optimising tasks for the first one, which is based on the "rule $n-1$" we ignore that the rule causes a perceptible increase of the number of extracts X_t. But if we use the "rule $n-1$" to solve the second task, it immediately decreases the number of extracts.

Theorem 9. *The complexity of algorithms* A1 *and* A2 *is linear to the number of maximal cliques in the graph* $G(V, E)$.

Proof. Suppose that the graph G includes S maximal cliques and any maximal clique has an average number of nodes equal to A. According to Theorem 8, algorithms separate from the graph G S sequences of extracts with the average length of A. Suppose that in every extract there are A nodes on average. Thus, the whole complexity is equal to $T = S*A*A^2 = S*A^3$.

Increasing the number of maximal cliques in G (for example $S1 > S$), the complexity will be increased by $T1 - T = S1*A^3 - S*A^3 = A^3(S1 - S)$. Consequently, the complexity of algorithms A1 and A2 is linear to the number of maximal cliques in G.

The theorem is proved.

Theorem 9 is based on the assumption that for every maximal clique its own n-tree is formed. In fact, algorithms A1 and A2 expand this situation.

1. The leading node and exhaustive nodes are eliminated from X_t.
2. The number of nodes in $X_{t+1} \subset X_t$ is less at least by one than in X_t.
3. Where possible, the "rule $n-1$" is used.

6.1. Algorithm complexity on Moon-Moser graphs

Next, we show the complexity of algorithms A1 and A2 on the well-known Moon-and-Moser graphs (MM-graphs).

As known, a MM-graph G consists of groups of nodes so that the nodes of a group are not adjacent together, but they are adjacent to all other nodes of G.

Moon and Moser [6] proved that by adding new groups of nodes to the MM-graph, the number of maximal cliques in it increases exponentially.

Suppose that the MM-graph with R groups of nodes, three nodes in each, is given. Then it includes 3^R maximal cliques of R nodes.

Using our algorithms to extract all maximal cliques we do $\Sigma_j 3^j$ extracts, $j = 1,2,..., R-1$. Suppose that every extract has R nodes on average. As we

use adjacency matrix instead of the graph, the complexity is equal to $T = R^2 * \Sigma_j 3^j \approx 0.5 * R^2 * 3^R$.

7. OPTIMISATION OF ALGORITHMS A1 AND A2

It is clear that the clique originality testing is needed because of the adequacy missing between the node weight and eliminations in algorithms A1 and A2. It means that there can exist situations where all maximal cliques that include a new node have already been extracted (we do not know this!), but the node has non-zero weight in X_0.

This conflict is derived from the essence of these algorithms. They are greedy algorithms by which all edges between the node Y_t and its neighbours in extraction X_t are eliminated from X_t. A1 and A2 do not take into consideration the situation, when only the elimination of single edges is needed. A contradictory situation arises here and to remove it, the testing described is needed.

Elimination of a node means the elimination of all edges between the node and its neighbours. In A1 and A2 we do it on every level t, $t \geq 0$, related to level $t+1$ eliminating the leading node and exhaustive nodes from X_t. But it has no feedback always to levels $t-1$, $t-2$, etc.

The question arises here: when and how can we eliminate single edges?

The elimination with feedback would be essentially global, it means that the elimination of edges proceeds from the structure of the graph G, not from any subgraph $G' \subset G$. Ignoring it, for special testing in algorithms A1 and A2, step P9 is needed.

This conflict is not eliminated by using the terms "feasible" and "banned" node (see Def. 4). The nodes may be banned one relative to another, it means they have no feasible neighbours, but we have not yet analysed them by three, by four, etc. together.

The adjacency matrix establishes adjacency between the nodes. It means that the weight of the node equals to its degree in the adjacency matrix. To give the correct answer to our question, for every adjacent node pair, we have to estimate the existence of the common node in addition to the nodes of the leading nodes sequence. If they have no more common nodes, we may eliminate the edge between those two nodes from several levels $< t$. We can do the process of elimination of an edge only related to the nodes for which all cliques are not yet extracted, because all edges for the nodes with all extracted cliques are also eliminated.

Thus, the edge between the adjacent node pair in X_t, which has no common adjacent node with the described property, can be eliminated from several levels $< t$.

To realise this approach, we have to define some new terms.

Definition 6. *The node relative to which all maximal cliques have been extracted is called an exhausted node. All the other nodes are called free nodes.*

Thus, differently from the banned nodes, an exhausted node is a node which on level 0 has been

1) chosen to be a leading or

2) an exhaustive node.

Definition 7. *The free node, which has not been chosen to be a member of the current leading nodes sequence and does not belong to the X_t, is called a decision node on level t.*

We need the term to explain which edges and when can be eliminated.

For all of the theorems below, we suppose that the extracting of repetitive cliques is excluded. It means that in this situation, the special testing is used (step P9).

Theorem 10. *The edge between the arbitrary nodes A and B of X_t can be eliminated from X_{t-1} only if no common decision node for A and B among the nodes of X_{t-1} exist.*

Proof. We shall show that a conflict arises if

1) the decision node exists and the edge is eliminated,

2) the decision node does not exist and the node is not eliminated.

1. Suppose by contradiction that we can eliminate the edge if a decision node exists in the extraction X_{t-1}. Then the elimination of the edge on level $t-1$ means that on back-tracking to the level $t-1$, doing extraction by this decision node, there exists no edge between A and B. Since the decision node has to belong to X_{t-1}, then not all maximal cliques have been extracted yet relative to it. As a result of the elimination of this edge, a situation is created where the maximal clique, which includes that edge, will be unextracted. That is in conflict with the assumption that all maximal cliques will be extracted.

2. Suppose that a decision node does not exist and we do not eliminate the edge between A and B. If there is no decision node among the nodes of X_{t-1}, then this pair of nodes (A, B) could not be obtained relative to any node of X_{t-1}. But it means that relative to X_{t-1} (it means relative to the current sequence of leading nodes $Y_0, Y_1, \ldots, Y_{t-2}$) for this pair (A, B) all maximal cliques have been extracted. If we do not eliminate this edge, then on back-tracking to the level $t-1$, one of these nodes, A or B will be chosen as the leading node. A conflict has been created which can cause the repetitive clique extraction.

The theorem is proved.

According to this theorem, we have to estimate all pairs of connected nodes on X_t and eliminate the edge between the nodes from the previous extraction X_{t-1},which has no common decision node in it.

This approach reduces the need to step P9 of algorithms A1 and A2. Algorithm A3 described below takes into consideration the results of Theorem 10. As in algorithms A1 and A2, in the algorithm A3 by the node weight we imply the node degree in an extracted subgraph.

7.1. Algorithm A3

P1. Initialising. $i := 0$, $\text{CLIQUE}_0 := \{\}$, $F_0 := \{\}$.

P2. Calculation of weights of nodes of X_0 to F_0.

P3. Check of a maximal clique. IF the weight of the node(s) K in F_i is equal to zero, we have found a maximal clique(s). Output CLIQUE_i and the node K.

P3A. IF the weights of the nodes in X_i are equal to the number of nodes with non-zero weight -1 THEN BEGIN a maximal clique is found. Output all nodes with non-zero weight of X_i and CLIQUE_i as a maximal clique. Set all weights in F_i to zero END

P4. Banning of nodes and control for back-tracking. IF the clique(s) was found THEN ban its nodes and ban them relative to themselves. IF all weights in F_i are zeros THEN

P4A. BEGIN $i = i - 1$. IF $i = -1$ THEN GOTO FINISH ELSE GOTO P3A END

P5. Selection of the leading node. $\text{CLIQUE}_{i+1} := \text{CLIQUE}_i$. As a leading node, find a feasible node J with the maximal weight in F_i ($C = 1$). If there are no feasible nodes, find a node J with a maximal number of free neighbours ($C = 2$). If there are several nodes, take the node with the maximal weight. If there are no such nodes, take the first node with a maximal weight ($C = 3$). Add J to the CLIQUE_{i+1}.

P6. Elimination of node and recalculation of weights. Eliminate the node J and recalculate weights in F_i (set zero to the weight of node J in F_i and diminish its neighbours' weights by one).

P7. Formation of a new generation. $i := i + 1$. Exclude the submatrix X_i corresponding to the node J from X_{i-1}. Calculate the weights for F_i.

P8. The edge elimination control. IF there exists an edge (J,G) in X_i, which has no adjacent neighbours among free nodes of X_{i-1} THEN eliminate this edge (J,G) from X_{i-1} and decrease the frequency of the nodes J and G in F_{i-1} by one.

P9. Control of the originality of the extracted clique. IF $C < 3$ THEN GOTO P3. IF there exists a node which belongs to any extracted maximal clique, but does not belong to CLIQUE_i and X_i, and which is adjacent to all nodes in CLIQUE_i and X_i THEN GOTO P4A ELSE GOTO P3.

FINISH

On determining common neighbours of the connected node pairs in X_t, the maximal number of compared node pairs is equal to $(N_t - 1)(N_t - 2)/2$ in algorithm A3.

But there exists the second solution which matches with the technique of extractions used in A1 and A2. On this basis, the algorithm A1 has to complete the following activities: we choose the leading node Y_{t-1} and then determine the existence of common decision nodes between other nodes of the extraction X_t, based on Y_{t-1} with node Y_{t-1}. For these nodes A of X_t which have no common decision node with Y_{t-1}, we eliminate the edge (Y_{t-1}, A) from all extractions X_i, $i < t$.

Next, we will prove the validity of the theorem.

Theorem 11. *From all levels i, $i < t$, we can eliminate the edge between the leading node Y_{t-1} and the nodes A of extraction X_t, based on Y_{t-1}, if there exists the edge (Y_{t-1}, A), which has no common decision node on level t.*

Proof. According to the definition, the decision node on level t is a free node which neither belongs to X_t nor to the current sequence $Y_0, Y_1, \ldots, Y_{t-1}$. If the leading node Y_{t-1} and the arbitrary node A of X_t have no common decision node on level t, then it means that any graph $G_i \subset G$, $0 \leq i < t$, which belongs to the extraction X_i, based on the leading node Y_i of the current sequence, does not include any node relative to which we can analyse the connected node pair (Y_{t-1}, A). It means that all maximal cliques containing this edge are extracted. Consequently, we can eliminate this edge from all levels i, $i < t$.

The theorem is proved.

The difference between Theorems 10 and 11 in the context of algorithmics can be described as follows. In the first theorem, the edge is eliminated only from the previous extraction, in the second one, from all extractions before the last.

But what can we do in case of Theorem 11 with the node pair (Y_{t-1}, A), which has a decision node on level t $(i = t)$? Can we eliminate this edge or not? If it is possible, then from which extractions X_i, $i < t$?

The next theorem will give an answer to these questions.

Theorem 12. *The edge between the leading node Y_{t-1} and the node A of the X_t, extracted by Y_{t-1}, which has a common decision node U on level t, can be eliminated from all levels $t - i$, $0 < i < r$, where r is the number of the level on which the node U became the decision node.*

Proof. Suppose that the leading node Y_{t-1} and the node A from X_t extracted by Y_{t-1} has a common decision node U on level t. Suppose by contradiction that we do not eliminate this edge. If we extract a maximal clique and do back-tracking to level $t - 1$, then there are no conflicts because of elimination of Y_t from X_{t-1}, it means that all edges of the node Y_{t-1} are eliminated. Problems arise when we do back-tracking to the level $t - i$, $1 < i \leq t$. Then we have two possibilities.

1. This edge may fall into the new extraction $X_{t-2} \subset X_{t-1}$. That occurs when the node U is chosen to be the leading Y_{t-2}, on the basis of which the new extraction X_{t-1} will be made. In this case, the original maximal clique will be extracted.

2. If the node U does not belong to the X_{t-2}, then as the weight of the nodes Y_{t-1} and A is not equal to zero in X_{t-2} (we did not eliminate this edge (Y_{t-1}, A)!), any node of this pair would be chosen to be the leading node some time and then the extraction of unoriginal clique is done.

Thus, in the second case we have reached a conflict. This conflict preserves for all back-trackings to level $t-2$, $t-3$ and so on, while the situation described in the first case arises, i.e. until we have reached the level $t-r$ in extraction X_{t-r} of which the node U exists.

The theorem is proved.

The described approach is used in algorithm A4. In our comparison of algorithms A3 and A4 in case of an extraction X_t, we do only N_t comparisons. But when using the results of Theorems 11 and 12, we have to recalculate the weights of the nodes not only on level $t-1$ but on all levels i, $i < t$.

7.2. Algorithm A4

P1. Initialising. $i := 0$, $\text{CLIQUE}_0 := \{\}$, $F_0 := \{\}$.

P2. Calculation of weights of nodes of X_0 to F_0.

P3. Check of a maximal clique. IF the weight of some node(s) K in F_i is zero, we have found a maximal clique(s). Output CLIQUE_i and the node K.

P4. Control for back-tracking. IF all weights in F_i are zeros THEN

P4A. $i := i - 1$. IF $i = -1$ THEN GOTO FINISH. IF there are no nodes under analysis THEN GOTO P4A.

P5. Selection of the leading node. $\text{CLIQUE}_{i+1} := \text{CLIQUE}_i$. As a leading node, find a feasible node J with the maximal weight in F_i $(C = 1)$. If there are no feasible nodes, find a node J with a maximal number of feasible neighbours $(C = 2)$. If there are several nodes, take the node with the maximal weight. If there are no such nodes, take the first node with a maximal weight $(C = 3)$. Add J to CLIQUE_{i+1}.

P6. Formation of a new generation. $i := i + 1$. Exclude the submatrix X_i corresponding to the node J from X_{i-1}. Calculate the weights for F_i.

P7. The edge elimination control. IF there exists a node G which has no adjacency neighbours among free nodes with leading node J THEN eliminate this edge (J, G) from all X_q, $0 \le q < i$.

P8. IF there exists a neighbour R of such kind THEN eliminate edge (J, G) from all X_r, $r \le q < i$, where r is a number of level, from which R changed to the decision node.

P9. Control of the originality of the extracted clique. IF $C < 3$ THEN GOTO P3. IF there exists a node which belongs to any extracted maximal clique, but does not belong to CLIQUE$_i$ and X_i, and which is adjacent to all nodes in CLIQUE$_i$ and X_i THEN GOTO P4A ELSE GOTO P3.
FINISH

Algorithm A4 does not enable us to apply the "rule $n-1$" because the results of decision node determination for nodes of extraction X_t have to be resounded in previous extractions i, $i < t$. We can exclude this situation by using of the technique of algorithm A3. It means that for using the "rule $n-1$", we have to find decision nodes for all edges of X_t as we did in algorithm A3, next eliminate the required edges as it was required in A4 and then use the "rule $n-1$". Since we do that only once, at the end of current sequence of extractions, no conflicts arise.

8. HOW DO YOU GET RID OF TESTING OF CLIQUE REPETITIVENESS

If the clique to be extracted is not unique, we do superfluous extractions. As we saw, we have decreased the number of turns to the testing of originality of clique to be extracted, but we have not excluded that completely. It means that there exist situations where we do not know if we can eliminate the edge or not. These are situations when the nodes of the edge and their decision node were involved in the extracted maximal clique.

The next theorem gives an answer to the question in the sub-heading.

Theorem 13. *Suppose that we are on level t and treat the elimination of the edge* (A, B) *at which we have fixed all common free neighbours* $\{j\}$ *of A and B,* $\{j\} \neq \varnothing$. *We cannot eliminate the edge* (A, B) *from levels i,* $i < t$, *if in the graph* $G_j \subset G$, *described by the nodes of* $\{j\}$ *and their neighbours, there exists a maximal clique T,* $T \neq \{Y_0, Y_1, ..., Y_{t-1}\}$, *all nodes of which have no common exhausted node with A and B.*

Proof. By Definition 1 for the exhausted node, all maximal cliques have been extracted. If every maximal clique of the graph G_j has a common exhausted node with edge (A, B), it means that we have treated any association of maximal clique and edge (A, B). Thus there is no association of nodes relative to which we have not analysed the edge (A, B). Consequently, the nodes A and B will exhaust on level t relative to the sequence $Y_0, Y_1, ..., Y_{t-1}$ and we have to eliminate the edge (A, B) from all levels i, $i < t$.

If there exists a maximal clique T, which has no common exhausted node with edge (A, B), it means that this association of nodes (A and B and nodes of clique T) has not been treated earlier. Consequently, maximal

clique has not been extracted yet where these nodes exist together and we cannot eliminate this edge.

The theorem is proved.

As we see, the determination of the situation described by Theorem 13 is very labour-consuming since for every edge we would determine all maximal cliques in the worst case. Therefore the question of suitability arises here. The special testing (step P9 of algorithms) is more suitable for realisation.

9. PROBLEMS IN REALISATION OF ALGORITHMS A1 AND A2

The extract X_t includes nodes which are adjacent to every member of the sequence $Y_0, Y_1,..., Y_{t-1}$. One has to know the nodes of X_t to determine the nodes of X_{t+1} and to calculate their weights. Thus, it is unsuitable to remember the X_t.

To determine the nodes of level $t+1$, the set-theoretical operation "intersection" is suitable, i.e., intersection over the set of adjacency vectors of leading nodes in sequence. These vectors are bitvectors where "0" denotes "not adjacent", "1" denotes "adjacent". As the leading nodes are joined to the sequence one by one, we do not have to find the intersection over all the bitvectors. We can intersect only the vector of the last intersection with the vector of the last leading node. Thus, for the extract of X_t, we have to intersect only two bitvectors.

It gives us the following advantages.

1. The main techniques of algorithms A1 and A2 are preserved.

2. We can now fix the exhaustedness of the node on level t just by zeroing the corresponding bit value in the vector of intersections. Earlier we did it by nullifying the node's weight.

We can realise the technique of fixing the common nodes of leading nodes by using the lists. The choice of a suitable way is made by the programmer.

10. PROBLEMS OF REALISATION OF ALGORITHMS A3 AND A4

The technique used in algorithms A3 and A4 requires remembering of all extractions X_i of the current sequence of the leading nodes $Y_0,..., Y_i,...,Y_t$ or the whole set of eliminated edges. It is necessary because we cannot identify existing adjacencies between the nodes by their weights in X_t.

In this section, we present some recommendations for moderating of this shortcoming. It is the mixed technique of algorithms A1 and A3, which is based on A3, but has not to remember all extractions X_i of the current sequence of the leading nodes.

The main idea of this technique is as follows. In algorithm A3 by Theorem 10, the elimination of all these edges of X_t from X_{t-1} is done, which has no common decision node among the nodes of X_{t-1}. In A1, the testing is similar to that done by comparing node weights on levels t and $t-1$. All nodes with equal weights are eliminated from X_{t-1}. From A3, it corresponds to the situation where the node has no common decision node among the nodes of X_{t-1} with any of its connected neighbours in X_t. Thus, algorithm A1 does not fix the situations when relative to certain node of X_t we have to eliminate less edges from X_{t-1} than it has neighbours in X_t.

Thus, using the technique of A1, in A3 we have to remember only these X_t, in case of which the elimination of not all edges but only several ones was done from X_{t-1}. For other situations, it is necessary to remember only lists or bitvectors of nodes.

11. USE OF MINIMAL OR MAXIMAL WEIGHT

In algorithms A1 to A4 we have used only maximal weight when choosing the leading node.

Actually we may choose any node with non-zero weight as the leading, these algorithms still congregate. (When proving of the theorems, we supposed that!) But to accelerate this process, we may proceed from minimal or maximal weight. Use of the minimal or maximal weight depends on the graph density. Our practice has shown that maximal weight works better for densities less than 50%, minimal weight works better in the case of density more than 50%. If we use the maximal weight, then in general, the maximum clique is extracted among the first cliques, in the case of the minimal weight, it will happen within the last cliques.

Consequently, when using the maximal or minimal weight, the main steps of algorithms A1 to A4 (elimination, feedback, extraction, testing of originality and so on) do not change.

12. DISCUSSION

Algorithms A3 and A4 do not enable us to avoid completely the testing of originality of the clique to be extracted, it is the step P9 of algorithms A1 and A2, but they enable to decrease the number of turns to it.

To generalise the technique used in algorithms A1 to A4, we can state the following.

1. In the case of these algorithms, the number of extractions depends on the choice of the leading node and graph density.

2. For algorithms A1 to A3, the number of extractions can decrease by using "rule $n-1$".

3. Using algorithm A2, the number of extractions X_t in extracting all maximal cliques compared with algorithm A1 in the worst case is not greater than with A1. In case of algorithm A2, feasible node or the node with feasible neighbours in X_t has to be chosen as the leading one, exclusively. Ignoring that creates a situation extracting a repetitive clique which requires special testing (step P9).

4. Algorithms A3 and A4 differ in their process of elimination of the edge. In the case of A3, the recalculation of weights is made only on level $t-1$, in algorithm A4 on all levels i, $i<t$. A4 is better since we find the decision node only for edge (Y_t, A), where A belongs to X_{t+1} extracted by Y_t, in A3 for all edges (A, B), where A and B belong to X_{t+1}.

REFERENCES

1. Bron, C. and Kerbosch, J. Algorithm 457: Finding all cliques of an undirected graph. – Comm. ACM, 1973, **16**, 575–577.
2. Chiba, N. and Nishizeki, T. Arboricity and subgraph listing algorithms. – SIAM J. Comput., **14**, 1, February 1985, 210–223.
3. Муллат И. Экстремальные монотонные системы. – Автоматика и телетехника, 1976, 5, 130–139.
4. Kuusik, R. The super-fast algorithm of hierarchical clustering and the theory of monotone system. – Trans. Tallinn Techn. Univ., 1993, **734**, 37–62.
5. Võhandu, L. and Kuusik, R. Cliques and algorithms with a hidden parallelism. – Trans. Tallinn Techn. Univ., 1993, **734**, 63–74.
6. Moon, J. and Moser, L. On cliques in graphs. – Israel J. Math., 1965, 3, 23–28.

KÕIKIDE MAKSIMAALSETE KLIKKIDE ERALDAMINE: MONOTOONSETE SÜSTEEMIDE TEOORIA KÄSITLUS

Rein KUUSIK

Artiklis on käsitletud orienteerimata lõplikest graafidest kõikide maksimaalsete klikkide eraldamist monotoonsete süsteemide teooria seisukohast. See probleem on NP-keeruline, maailma parimate algoritmide keerukus on lineaarne graafi maksimaalsete klikkide arvusse.

On esitatud ja tõestatud monotoonsete süsteemide teoorial baseeruvad algoritmid, nn. MONS-algoritmid, vaadeldava probleemi lahendamiseks. Nende puhul on lähtutud graafi seosmaatriksist, millele on ehitatud monotoonne süsteem.

Kirjeldatud lähenemine võimaldas luua terve rea efektiivseid algoritme, mille keerukus on lineaarne graafi maksimaalsete klikkide suhtes. Kirjeldatud teooria ja meetod on graafist suurima kliki eraldamise efektiivsete algoritmide loomise alus.

ВЫДЕЛЕНИЕ ВСЕХ МАКСИМАЛЬНЫХ КЛИК С ИСПОЛЬЗОВАНИЕМ ТЕОРИИ МОНОТОННЫХ СИСТЕМ

Рейн КУУСИК

Рассмотрено применение теории монотонных систем при решении задачи выделения всех максимальных клик из конечных неориентированных графов. Описаны и доказаны соответствующие алгоритмы. Их сложность линейна числу максимальных клик в графе.