Asia Pac. J. Math. 2026 **13**:136

SOLVING ZERO-ONE LINEAR PROGRAMS USING WEIGHTED CLIQUES

SÁNDOR SZABÓ[1,*], BOGDÁN ZAVÁLNIJ[2]

[1]Department of Applied Mathematics, University of Pécs, Hungary
[2]HUN-REN Alfréd Rényi Institute of Mathematics, Hungary
[*]Corresponding author: sszabo7@hotmail.com

Received May 9, 2026

Abstract. The main result of this work is the observation that a maximization type 0-1 linear program with non-negative coefficients can be reduced to a finite number of maximum weight clique problems. To the given 0-1 linear program we tactically construct an auxiliary graph with weighted vertices and then we try to locate a maximum weight clique in the graph. The second observation is about the complexity of the algorithm. The fewer maximal cliques the auxiliary graph admits the higher is the efficiency of the proposed enumeration based algorithm. This observation can help to identify 0-1 linear programs that are computationally manageable.

2020 Mathematics Subject Classification: 05C15; 05B45; 52C22.

Key words and phrases. clique number; maximum clique; maximal clique; binary linear programming.

1. Introduction

The graphs in this work will have finitely many nodes and finitely many vertices, that is, we will work exclusively with finite graphs. Further, the graphs we consider will not have double edges connecting two nodes twice and the graphs will not have any loop connecting a node to itself. In short, in this paper we will deal with finite simple graphs. A list of the nodes and a list of the unordered pairs of the nodes that are connected by an edge are sufficient to describe a finite simple graph. A finite simple graph G can be identified with the ordered pair (V, E), where V is the set of nodes and E is the set of edges.

Let $G = (V, E)$ be a finite simple graph and let Δ be a subset of V. We say that Δ is a k-clique in G if $|\Delta| = k$ and for each $u, v \in V$, $u \neq v$ the nodes u, v are adjacent in G. (The unordered pair $\{u, v\}$ is an edge of G.) In the $k = 0$ case the set Δ is empty but we do not call the empty set a 0-clique. In the $k = 1$ case the set Δ has only one element. In this case we consider Δ to be a 1-clique. The 1-cliques are nodes, the 2-cliques are end points of edges. These are the trivial cliques in G.

DOI: 10.28924/APJM/13-136

For a finite simple graph G there is an integer k such that G contains a k-clique but G does not contain any $(k-1)$-clique. This well defined number k is called the clique number of G and it is denoted by $\omega(G)$. A k-clique with $k = \omega(G)$ is a called a maximum clique in G. A k-clique in G that cannot be extended to be a $(k+1)$-clique by adding a new node of G to it is called a maximal clique.

Problem 1. *Given a finite simple graph G and a positive integer k. Decide if G contains a k-clique.*

Problem 2. *Given a finite simple graph G. Determine $\omega(G)$.*

Problem 1 is referred to as the k-clique problem. Problem 2 is called the maximum clique problem. The k-clique problem is a decision problem and it belongs to the NP complete complexity class. The maximum clique problem is an optimization problem and it belongs to NP hard complexity class. For further details and applications of the k-clique and maximum clique problems see [2]. For complexity considerations see [3] or [6].

2. The restricted maximum clique problem

In multivariate calculus there are unconstrained and constrained optimization problems. We use Lagrange multipliers to reduce the constrained optimization problem to an unconstrained one. An analog situation occurs in the field of clique problems.

Problem 3. *Given a finite simple graph G and given a clique Ω in G. Find the size of a largest clique in G that does not contain Ω as a subclique.*

We call Problem 3 the constrained maximum clique problem. We call the clique Ω the prohibited clique in G. The purpose of this section is to show that the constrained maximum clique problem of the graph G can be reduced to an unconstrained maximum clique problem of a suitably constructed auxiliary graph G'.

First we describe the construction of the auxiliary graph G'. If the prohibited clique Ω is a 1-clique, then deleting Ω from G means deleting a node from G. The resulting graph will be the auxiliary graph G'. If the prohibited clique Ω is a 2-clique, then deleting Ω from G means deleting an edge from G. The resulting graph will be the auxiliary graph G'. In the remaining part of our argument we assume that the prohibited clique Ω has at least 3 nodes. The special case when the prohibited clique Ω is identical to the given graph G is again trivial. We may delete any node of G to get the auxiliary graph G'. Therefore for the remaining part we assume that the given graph G has more nodes than the prohibited clique Ω does.

Let V be the set of nodes of the given graph G. Suppose that the prohibited clique Ω has s nodes and let W be the set of nodes of Ω. (Keep in mind that $s \geq 3$ is assumed.) We choose a node w of Ω. We call w the pivot element of the construction. We partition the set of nodes V of G into three subsets A,B,C

such that $A = \{w\}$, $B = W \setminus \{w\}$, $C = V \setminus W$. For the sake of notational simplicity we assume that $V = \{1, \ldots, n\}$. (Remember that we have assumed that the inequality $s < n$ holds.) We may assume that $W = \{1, \ldots, s\}$ since this is only a matter of renaming the nodes of G. Further, we assume that the pivot element w is equal to 1. In this situation $A = \{1\}$, $B = \{2, \ldots, s\}$, $C = \{s+1, \ldots, n\}$. The node set V' of the constructed graph G' will be partitioned into three subsets A',B',C'. The elements of A' will be $1_2, \ldots, 1_s$, that is, A' will have $s-1$ elements. We set the subsets B',C' to be equal to B,C, respectively. In notations $A' = \{1_2, \ldots, 1_s\}$, $G' = B$, $C' = C$. In plain English, we remove node 1 from G and add $s-1$ new nodes $1_2, \ldots, 1_s$ to G when we construct the new graph G' from G. Clearly, $V' = \{1_2, \ldots, 1_s, 2, \ldots, n\}$. The nodes of the set $V' \setminus A' = \{2, \ldots, n\}$ will be adjacent in the new graph G' exactly in the same way as they were adjacent in the original graph G. The nodes $1_2, \ldots, 1_s$ will be pairwise non-adjacent in G'. The node 1_v can have neighbors in the graph G' in the sets B' and C'. The node 1_v will be connected by an edge in G' with each of the nodes $2, \ldots, s$ except node v. Further, node 1_v will be adjacent in G' to each node c in C' whenever node 1 was adjacent to c in C in the original graph G.

Let $\widehat{\omega}(G)$ be the maximum size among the cliques in the graph G that does not contain the prohibited clique Ω and let $\widehat{\omega}(G')$ be the maximum size among the cliques in the graph G'.

Lemma 1. *Using the notations above the equation $\widehat{\omega}(G) = \omega(G')$ holds.*

Proof. We will show that the inequalities $\widehat{\omega}(G) \leq \omega(G')$, $\omega(G') \leq \widehat{\omega}(G)$ hold.

In order to prove that $\widehat{\omega}(G) \leq \omega(G')$ set $\widehat{\omega}(G) = t$. The graph G contains a t-clique Δ such that Δ does not contain the prohibited clique Ω as a subgraph. We distinguish two cases depending on that the node 1 is a node of Δ or not. Assume first that node 1 is not a node of the t-clique Δ. It means that each node of Δ is an element of $B \cup C$. Using the fact that $B \cup C = B' \cup C'$ we draw the conclusion that the graph G' contains a t-clique Δ'. This implies that $t \leq \omega(G')$, that is, $\widehat{\omega}(G) \leq \omega(G')$.

Next assume that node 1 is a node of the t-clique Δ'. As Δ' does not contain the prohibited clique Ω as a subgraph there is an element of B which is not a node of Δ'. We may assume that node s is not a node of Δ' since this is only a matter of renaming the nodes of G. In this situation G' contains a $(t-1)$-clique Δ'' whose nodes are in the set $(B' \cup C') \setminus \{1, s\}$. Augmenting the $(t-1)$-clique Δ'' with node 1_s we get a t-clique Δ^* in G'. This implies that $t \leq \omega(G')$, that is, $\widehat{\omega}(G) \leq \omega(G')$.

In order to verify that $\omega(G') \leq \widehat{\omega}(G)$ we set $\omega(G') = t$. The graph G' contains a t-clique Δ'. We distinguish two cases depending on Δ' has a node in the set A' or not. Assume first that Δ' does not have any node in A'. Each node of Δ' is in $B' \cup C'$. There is a t-clique Δ^* in G whose nodes are in $B \cup C$. Plainly, Δ^* cannot contain the prohibited clique Ω as a subgraph since node 1 is not a node of Δ^*. This implies that $t \leq \widehat{\omega}(G)$, that is, $\omega(G') \leq \widehat{\omega}(G)$.

Next assume that Δ' has a node in A'. The nodes in A' are pairwise non-adjacent and so Δ' contains exactly one element of A', say 1_s is a node of Δ'. (Please remember that node 1_s is not adjacent to node s in G'.) There is a t-clique Δ^* in G whose nodes are in $B \cup C$. Plainly, Δ^* cannot contain the prohibited clique Ω as a subgraph since node s is not a node of Δ^*. This implies that $t \leq \widehat{\omega}(G)$, that is, $\omega(G') \leq \widehat{\omega}(G)$. □

The alert reader will notice that Lemma 1 holds for vertex weighted graphs as well. In this more general setting to each node of the finite simple graph G a non-negative weight is assigned. To each clique Δ in G the total sum of the weights of the nodes of Δ is assigned. We can call this quantity the weight of the clique Δ. When we construct the new auxiliary graph G' from G, the nodes of the new graph G' inherit the weights from G. When we introduce new vertices $1_2, \ldots, 1_s$ we will assign the weight of the pivot node 1 to each of these new nodes.

3. The first example

In order to illustrate the procedure of constructing the auxiliary graph G' from the given graph G we worked out a small size example in details.

Example 1. *Let $G = (V, E)$ be a finite simple graph whose adjacency matrix is given in Table 1. The graph G has 8 nodes and 14 edges. The nodes of G are denoted by $1, \ldots, 8$, that is, $V = \{1, \ldots, 8\}$. The nodes of the prohibited 4-clique Ω are $1, 2, 3, 4$, that is, $W = \{1, 2, 3, 4\}$.*

Figure 1 exhibits a possible geometric representation of the given graph G. Figure 2 is a redrawing of G showing how the nodes are sorted into the subsets A,B,C. From Figure 3 we can see that the auxiliary graph G' has 10 vertices and 19 edges.

Table 1. The adjacency matrix of the graph G in Example 1.

	1	2	3	4	5	6	7	8
1	×	•	•	•	•			
2	•	×	•	•			•	
3	•	•	×	•				•
4	•	•	•	×		•		
5	•				×	•		•
6				•	•	×	•	
7		•				•	×	•
8			•		•		•	×

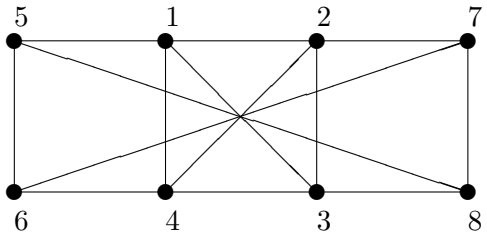

FIGURE 1. A possible geometric representation of the graph G in Example 1.

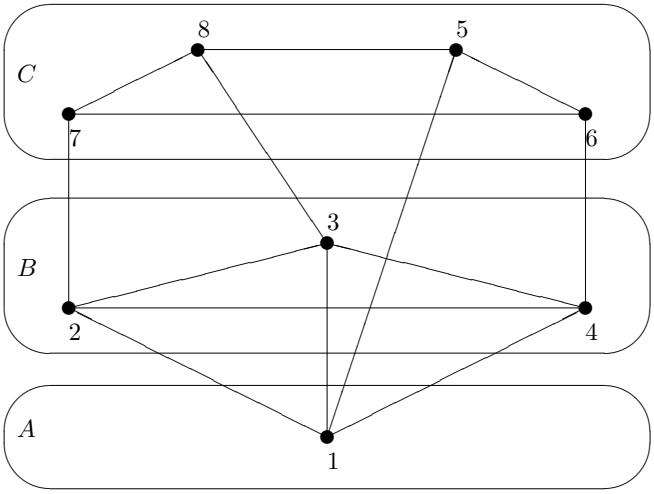

FIGURE 2. The nodes of the graph G in Example 1 are partitioned into subsets A,B,C.

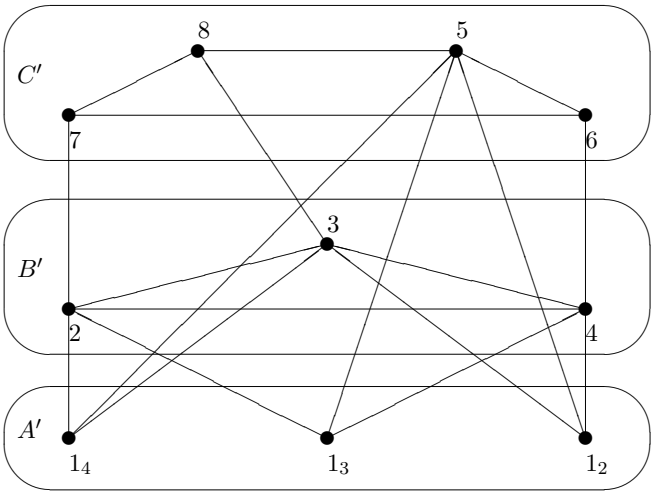

FIGURE 3. The nodes of the auxiliary graph G' in Example 1 are partitioned into subsets A',B',C'.

TABLE 2. The adjacency matrix of the auxiliary graph G' in Example 1.

	2	3	4	5	6	7	8	1_2	1_3	1_4
2	×	•	•			•			•	•
3	•	×	•				•	•		•
4	•	•	×		•			•	•	
5				×	•		•	•	•	•
6			•	•	×	•				
7	•				•	×	•			
8		•		•		•	×			
1_2		•	•	•				×		
1_3	•		•	•					×	
1_4	•	•		•						×

There is a mechanical way to construct the adjacency matrix of the auxiliary graph G' from the adjacency matrix of the given graph G. We carry out this construction in connection with the graph G in our example.

We start with the adjacency matrix of G. (It is given in Table 1.) The nodes of the prohibited clique Ω are 1,2,3,4. We choose node 1 to be the pivot element of the transformation. We add three new rows and columns to the adjacency matrix of G. These new rows and columns correspond to the new nodes 1_2,1_3,1_4.

We fill up complete rows in the new columns with circles. We will turn these circles into bullets later. What we do is nothing else than adding the edges of a complete bipartite graph to G. One vertex set of this bipartite graph consists of the neighbors of the pivot element. (In our case these nodes are 2,3,4,5.) The elements of the other vertex set of the bipartite graph are the new nodes. (In our case these nodes are 1_2,1_3,1_4.) In order to keep the constructed adjacency matrix symmetric with respect to the main diagonal, we fill complete columns in the new rows as well.)

Next, we cross out circles corresponding to the fact that node 1_v is not adjacent to node v for each node v of the prohibited clique Ω. Finally, we delete the row and the column of the pivot element.

TABLE 3. The construction of the adjacency matrix of the auxiliary graph G' in Example 1.

	1	2	3	4	5	6	7	8	1_2	1_3	1_4
1	×	●	●	●	●						
2	●	×	●	●			●		⊠	○	○
3	●	●	×	●				●	○	⊠	○
4	●	●	●	×		●			○	○	⊠
5	●				×	●		●	○	○	○
6				●	●	×	●				
7		●				●	×	●			
8			●		●		●	×			
1_2		⊠	○	○	○				×		
1_3		○	⊠	○	○					×	
1_4		○	○	⊠	○						×

4. DOMINANCE

We introduce a dominance relation on the set of vertices and on the set of edges of a finite simple graph. The idea of this dominance relation can be traced back to [4]. The vertex u dominates the vertex v in the finite simple graph G if the next conditions are met.

(1) u and v are distinct vertices in G.

(2) u and v are not adjacent in G.

(3) $N(v) \subseteq N(u)$.

Here $N(u)$ is the set of neighbors of u in G, that is,

$$N(u) = \{x : \ x \text{ and } u \text{ are adjacent in } G\}.$$

We say that the edge $\{u, v\}$ of G dominates the edge $\{v, w\}$ of G in G if the conditions below are satisfied.

(1) u,v,w are pair-wise distinct vertices in G.

(2) u and w are not adjacent in G.

(3) $[N(w) \cap N(v)] \subseteq [N(v) \cap N(u)]$.

The following result is proved in [7].

Lemma 2. *Dominated vertices and edges may be deleted from the graph when we are looking for a k-clique. If nodes mutually dominate each other, then we retain at least one of these nodes. Similarly, if edges mutually dominate each other, then we retain at least one of these edges.*

Lemma 2 is valid not only for the case of k-cliques but it is valid for the case of maximum cliques as well. Moreover, the dominance relation can be extended for finite simple graphs with weighted nodes. The extra condition we should check is if the weight assigned to node u is not smaller than the weight assigned to node v in the definition of the vertex dominance. The additional condition we should check is if the weight assigned to node u is not smaller than the weight assigned to node w in the definition of the edge dominance.

The reader can verify that Lemma 2 holds for finite simple graphs with weighted nodes. Lemma 2 suggests tidying up the given graph by deleting dominated nodes and edges before feeding the graph into a maximum weight clique solver.

The node and vertex dominance can be generalized. For a clique Δ in G we define the neighborhood of Δ by

$$N(\Delta) = \bigcap_{v \in \Delta} N(v).$$

We say that the clique Δ dominates the clique Ω in the vertex weighted finite simple graph G if the following conditions met.

(1) $\Omega \cap [N(\Delta)] = \emptyset$.

(2) The weight of Δ is not smaller than the weight of Ω.

(3) $N(\Omega) \subseteq N(\Delta)$.

The extended version of Lemma 2 is the following.

Lemma 3. *A dominated clique may be prohibited from the vertex weighted graph G when we are looking for a maximum weight clique in G. If cliques are dominating each other mutually, then please make sure that at least one of the cliques is not prohibited.*

Note that locating a dominated clique leads to a prohibited clique and so to a restricted maximum weight clique problem. This in turn can be reduced to an unconstrained maximum weight clique problem.

5. The auxiliary graph

Let us consider the following 0-1 linear program PROG. Maximize the objective function

$$\sum_{j=1}^{n} c_j x_j \tag{1}$$

subject to the constraints

$$\sum_{j=1}^{n} a_{i,j}x_j \leq b_i, \tag{2}$$

for each i, $1 \leq i \leq m$. We assume that $0 \leq a_{i,j}$, $0 \leq b_i$, $0 \leq c_j$ for each i, j, $1 \leq i \leq m$, $1 \leq j \leq n$.

If $a_{i,j} > b_i$ holds for some i, j, $1 \leq i \leq m$, $1 \leq j \leq n$, then $x_j = 0$ must hold for any feasible solution of PROG. In other words, the value of the variable x_j must be fixed on level 0. This reduces PROG to a new 0-1 linear program which has fewer variables. We assume that we have carried out an inspection and have eliminated all the variables that must be fixed on level 0.

We assign a finite simple graph G to the 0-1 linear program PROG. The nodes of G are $1, \ldots, n$. Two distinct nodes u,v are not adjacent in G if the inequality $a_{i,u}x_u + a_{i,v}x_v > b_i$ holds for some i, $1 \leq i \leq m$. To the vertex v of the graph G we assign the non-negative weight c_v. The vertex v is associated with the variable x_v of the linear program and the weight of the node is the coefficient of x_v in the objective function. We call the graph G the conflict graph or auxiliary graph assigned to the given linear program.

For the sake of easier reference we state some straight-forward connections between the linear program PROG and the conflict graph G as lemmas.

Lemma 4. *If U is the weight of a maximum weight clique in G, then U is an upper bound of the values of the objective function of PROG.*

Proof. The result of the lemma can be rephrased such that if L is the optimal value of the objective function of PROG, then L is a lower bound of the weight of the maximum weight clique in the conflict graph G.

Let E be the edge set of the conflict graph G. Note that adding the constraints

$$x_u + x_v \leq 1, \tag{3}$$

for each $\{u, v\} \notin E$ to the 0-1 linear program PROG the set of feasible solutions is not changing. Dropping the constraints (2) gives a new 0-1 linear program PROG2. PROG2 is nothing else than a possible linear programming reformulation of the maximum weight clique problem involving the conflict graph G. The set of feasible solutions of PROG2 contains the set of feasible solutions of PROG. Therefore the optimal value of the objective function of PROG is a lower bound of the optimal value of the objective function of PROG2. □

Lemma 5. *If v is a vertex of the conflict graph G, then the assignment $x_v = 1$, $x_j = 0$ for each j, $j \neq v$, $1 \leq j \leq n$ is a feasible solution the 0-1 linear program PROG.*

If $\{u, v\}$ is an edge of the of the conflict graph G, then the assignment $x_u = x_v = 1$, $x_j = 0$ for each j, $j \neq u$, $j \neq v$, $1 \leq j \leq n$ is a feasible solution of the 0-1 linear program PROG.

Proof. If the inequality $a_{i,v}x_v > b_i$ holds for some i, $1 \leq i \leq m$, then v cannot be a node of G. Thus if v is a node of G, then the assignment $x_v = 1$, $x_i = 0$ for each i, $i \neq v$, $1 \leq i \leq n$ is a feasible solution of the 0-1 linear program PROG.

If the inequality $a_{i,u}x_u + a_{i,v}x_v > b_i$ holds for some i, $1 \leq i \leq m$, then the unordered pair $\{u, v\}$ cannot be an edge of G. Thus if $\{u, v\}$ is an edge of G, then the assignment $x_u = x_v = 1$, $x_j = 0$ for each j, $j \neq u$, $j \neq v$, $1 \leq j \leq n$ is a feasible solution of the 0-1 linear program PROG. □

Going through the nodes of G we can locate a maximum weight 1-clique in G. Similarly, going through the edges of G we can locate a maximum weight 2-clique in G.

Lemma 6. *Let L be the largest among the weights of the 1-cliques and 2-cliques in G. This L is a lower bound of the values of the objective function PROG.*

Let G' be the graph we get from G by deleting the maximal 1-cliques and the maximal 2-cliques retaining only one 1-clique or one 2-clique whose weight is equal to L. The weights of the maximum cliques in G and in G' are equal.

Proof. By Lemma 5, the nodes and edges of G provide feasible solutions of PROG and so they provides lower bounds for the values of the objective function. This proves the first statement of the lemma.

A maximal 1-clique cannot be extended to a 2-clique in G by adding an extra node to it. Similarly, a 2-clique cannot be extended to a 3-clique in G by adding an extra node to it. Although the graph G' may have fewer nodes and edges than G we cannot loose all the maximum weight cliques appearing in G when we reduce G to G'. □

We propose the following algorithm to solve the given 0-1 linear program PROG (1), (2). Let us construct the auxiliary graph G. Feeding G into a standard maximum weight clique solver we can locate a maximum weight clique Δ in G. (A possible maximum weight clique solver can be found in [5]. Unfortunately, it is a completely realistic scenario that the graph G is outside of the range of feasibility of the used solver.) The weight of the clique Δ provides an upper bound for the objective function of PROG. The nodes of Δ specifies an assignment of 0-1 values to the variables $x_1, \ldots, x_n$. If this assignment is a feasible solution of PROG, then we have located an optimal solution of PROG. If the assignment is not a feasible solution of PROG, then we prohibit the clique Δ from G. At this stage we construct a new auxiliary graph G' form G and work with the graph G' in a similar way we have worked with G.

The maximum weight clique problem is a computationally demanding problem. It is just aggravating the difficulties that we are facing the solution of a sequence of maximum weight clique problems. However, if the auxiliary graph contains a manageable number of maximal cliques, then even an exhaustive search can solve the 0-1 linear program PROG. The point is that at this time not the number of the variables or the number of the constraints of the given 0-1 linear program are the decisive factors

of the complexity of the algorithm. A classical algorithm to enumerate all maximal cliques in a given finite simple graph is presented in [1].

6. The second example

We will illustrate the construction of the conflict graph assigned to a given 0-1 linear program. Then we will show how the conflict graph can provide information about the program.

Example 2. *Let us consider the* 0-1 *linear program which has* 10 *variables and* 2 *constraints. The coefficients are listed in Table* 4.

We have borrowed the 0-1 linear program in Example 2 from the book [8]. Namely, it appears as Example 4.6 in the book.

We rearranged the coefficients of the constraint $(1')$ into a decreasing sequence. The sequence starts with a largest coefficient and ends with a smallest. Table 5 exhibits the $\{x_u, x_v\}$ pairs for which the assignment $x_u = x_v = 1$ breaks constraint $(1')$. The number of these pairs is 29. Similarly, Table 6 lists the $\{x_u, x_v\}$ pairs for which the assignment $x_u = x_v = 1$ breaks constraint $(2')$. The number of these pairs is 20. (The two lists of the pairs are overlapping.) An assignment $x_u = x_v = 1$ which is compatible with constraint $(1')$ and constraint $(2')$ provides an edge $\{u, v\}$ of the conflict or auxiliary graph G. The adjacency matrix of G can be seen in Table 7. A possible geometric representation of G can be seen in Figure 4.

Inspecting the vertices of G, we can see that among the 1-cliques of G the 1-clique $\{9\}$ has the largest weight. Namely, this weight is equal to 50. Inspecting the edges of G, we find that among the 2-cliques of G the 2-clique $\{1, 10\}$ has the largest weight. This weight is equal to $21 + 47 = 68$. By Lemma 5, we have located a feasible solution of the program and we have located a lower bound for the value of the objective function. In particular, 68 is a lower bound for the value of the objective function.

The nodes 7 and 8 are isolated nodes of G and so the 1-cliques $\{7\}$, $\{8\}$ are maximal cliques of G. The degrees of the nodes 5 and 9 are equal to 1 and consequently the 2-cliques $\{4, 5\}$, $\{4, 9\}$ are maximal cliques of G. By Lemma 6, we may delete the nodes 7,8 and the edges $\{4, 5\}$, $\{4, 9\}$ from the graph G. After deleting the two edges the nodes 5,9 became isolated and so we may delete these nodes as well.

Node 10 dominates nodes 2,3,6,7 and so nodes 2,3,6, may be deleted from G when we are looking for a maximum weight clique in G. With this move graph G reduces to the 3-clique $\{1, 4, 10\}$. This is a maximum weight clique in G with weight 83 and so 83 is an upper bound for the value of the objective function of the given 0-1 linear program. However, the assignment $x_1 = x_4 = x_{10} = 1$ breaks constraint $(1')$. Therefore, the 3-clique $\Omega = \{1, 4, 10\}$ should be prohibited from the conflict graph G. We are facing the situation when the prohibited clique Ω is identical with the auxiliary graph G. As node 4 has the smallest weight in G, we delete the node 4 from G to get the new auxiliary graph G'. Clearly,

the 2-clique $\{1, 10\}$ has the largest weight in G'. The weight of this clique is equal to 68. It means that 68 is an upper bound for the values of the objective function of the given 0-1 linear program. As the upper and the lower bounds of the objective function are equal we have located the optimum value of the objective function.

We state the final conclusion. The optimum value of the given 0-1 linear program is equal to 68. The assignment $x_1 = x_{10} = 1$, $x_2 = \cdots = x_9 = 0$ is a possible optimal solution.

TABLE 4. The coefficients of the 0-1 linear program in Example 2.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}		
$(1')$	20	34	42	11	48	39	53	23	46	38	$\leq$	63
$(2')$	19	35	23	23	39	35	45	61	47	33	$\leq$	72
	21	32	45	15	19	42	15	37	50	47	$\rightarrow$	max

TABLE 5. The list of the 29 assignments $x_u = x_v = 1$ that break constraint $(1')$ in Example 2.

	x_7	x_5	x_9	x_3	x_6	x_{10}	x_2	x_8	x_1	x_4		
	53	48	46	42	39	38	34	23	20	11	$\leq$	63
1	×	×										
2	×		×									
3	×			×								
4	×				×							
5	×					×						
6	×						×					
7	×							×				
8	×								×			
9	×									×		
10		×	×									
11		×		×								
12		×			×							
13		×				×						
14		×					×					
15		×						×				
16		×							×			
17			×	×								
18			×		×							
19			×			×						
20			×				×					
21			×					×				
22			×						×			
23				×	×							
24				×		×						
25				×			×					
26				×				×				
27					×	×						
28					×		×					
29						×	×					

TABLE 6. The list of the 20 assignments $x_u = x_v = 1$ that break constraint $(2')$ in Example 2.

	x_8	x_9	x_7	x_5	x_6	x_2	x_{10}	x_3	x_4	x_1		
	61	47	45	39	35	35	33	23	23	19	$\leq$	72
1	×	×										
2	×		×									
3	×			×								
4	×				×							
5	×					×						
6	×						×					
7	×							×				
8	×								×			
9	×									×		
10		×	×									
11		×		×								
12		×			×							
13		×				×						
14		×					×					
15			×	×								
16			×		×							
17			×			×						
18			×				×					
19				×	×							
20				×		×						

TABLE 7. The adjacency matrix of the auxiliary graph G in Example 2.

										1
	1	2	3	4	5	6	7	8	9	0
1	×	•	•	•		•				•
2	•	×		•						
3	•		×	•						
4	•	•	•	×	•	•			•	•
5				•	×					
6	•			•		×				
7							×			
8								×		
9				•					×	
10	•			•						×

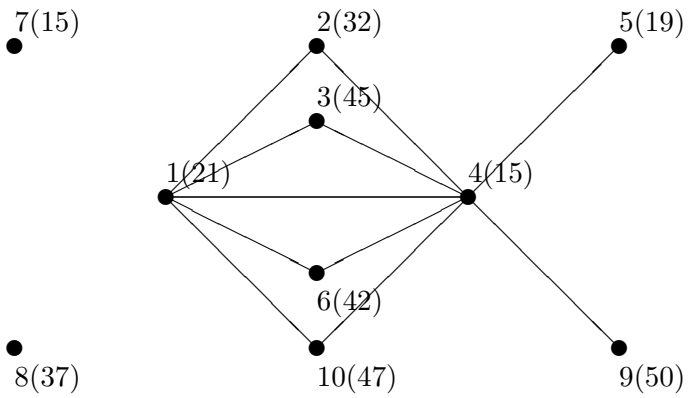

FIGURE 4. A possible geometric representation of the graph G in Example 2. The number near to a point is the name of the node. The number in parentheses is the weight of the node.

Authors' Contributions. All authors have read and approved the final version of the manuscript. The authors contributed equally to this work.

Conflicts of Interest. The authors declare that there are no conflicts of interest regarding the publication of this paper.

REFERENCES

[1] C. Bron, J. Kerbosch, Algorithm 457: Finding All Cliques of an Undirected Graph, Commun. ACM 16 (1973), 575–577. https://doi.org/10.1145/362342.362367.

[2] R. Carraghan, P.M. Pardalos, An Exact Algorithm for the Maximum Clique Problem, Oper. Res. Lett. 9 (1990), 375–382. https://doi.org/10.1016/0167-6377(90)90057-C.

[3] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, 1979.

[4] L. Butz, P.L. Hammer, D. Haussmann, Reduction Methods for the Vertex Packing Problem, in: Probability Theory, Proceedings of the 7th Conference, VNU Science Press, 1984, pp. 73–79.

[5] P.R.J. Östergård, A New Algorithm for the Maximum-Weight Clique Problem, Nord. J. Comput. 8 (2001), 424–436.

[6] C.H. Papadimitriou, Computational Complexity, Addison-Wesley Publishing Company, 1994.

[7] S. Szabó, Parallel Algorithms for Finding Cliques in a Graph, J. Phys.: Conf. Ser. 268 (2011), 012030. https://doi.org/10.1088/1742-6596/268/1/012030.

[8] H.P. Williams, Logic and Integer Programming, Springer US, 2009. https://doi.org/10.1007/978-0-387-92280-5.