Applied Intelligence
https://doi.org/10.1007/s10489-022-04304-7

A stepped tabu search method for the clique partitioning problem

Joaquín A. Pacheco[1] · Silvia Casado[1]

Accepted: 27 October 2022

Abstract
Given an undirected graph, a clique is a subset of vertices in which the induced subgraph is complete; that is, all pairs of vertices of this subset are adjacent. Clique problems in graphs are very important due to their numerous applications. One of these problems is the *clique partitioning problem* (CPP), which consists of dividing the set of vertices of a graph into the smallest number of cliques possible. The CPP is an NP-hard problem with many application fields (timetabling, manufacturing, scheduling, telecommunications, etc.). Despite its great applicability, few recent studies have focused on proposing specific resolution methods for the CPP. This article presents a resolution method that combines multistart strategies with tabu search. The most novel characteristic of our method is that it allows unfeasible solutions to be visited, which facilitates exploration of the solution space. The computational tests show that our method performs better than previous methods proposed for this problem. In fact, our method strictly improves the results of these methods in most of the instances considered while requiring less computation time.

Keywords Clique partitioning problem · Metaheuristics · Tabu search · Multistart methods

1 Introduction

Given an undirected graph $G = (V, E)$, where V represents the set of vertices and E represents the set of edges, a clique is a subset of V constituted by vertices that are pairwise adjacent – that is, for every pair of vertices i and j in such a subset, $(i, j) \in E$. A clique partition of graph G is a partition of V such that every subset in the partition is a clique. This work focuses on the problem of finding a clique partition of a graph G with the minimum cardinality. This problem is known as the *clique partitioning problem* (CPP) or the *minimum clique partition problem*, and it is an NP-hard problem [19]. Figure 1 of Section 2 shows an example of an undirected graph with 9 vertices, and Fig. 6 presents an optimal solution to this problem with three cliques.

The problem described above should be distinguished from another problem of the same name, which was addressed in Lü et al. [12] and Hu et al. [8] among other references. This problem is defined on complete graphs with weights on the edges. The aim is to divide the set of vertices into subsets such that the sum of the weights of the induced subgraphs is as small as possible.

Clique problems are very popular in the literature on graph problems. Perhaps the most widely known is the *maximum clique problem* (MCP), which consists of searching in G for a maximum clique – that is, a clique of maximum cardinality. Another commonly used concept is the maximal clique, which is a clique that is not contained in another clique. A maximum clique must be maximal, but a maximal clique does not have to be maximum. Pardalos and Xue [15] performed a thorough analysis of the *maximum clique problem* (considering formulations, complexity, algorithms, etc.), and Wu and Hao [21] conducted a very complete survey of this problem. Other important works are those of San Segundo and Artieda [16] that shows a practical application related to unmanned autonomous vehicles and San Segundo et al. [17], a methodological paper proposing improvements in the exact algorithms for this problem. Other known problems regarding cliques are *the maximal clique problem* [9], *the clique enumeration problem* [5] and *the clique coloring problem* [11].

Regarding the CPP, several applications can be found in different areas, such as airport logistics [1, 6], timetabling [13, 20], manufacturing [4], the register-transfer synthesis of data paths [10], social networks and the internet [18], telecommunications [22], and data science [3, 14].

✉ Joaquín A. Pacheco
jpacheco@ubu.es

Silvia Casado
scasado@ubu.es

[1] University of Burgos, Burgos, Spain

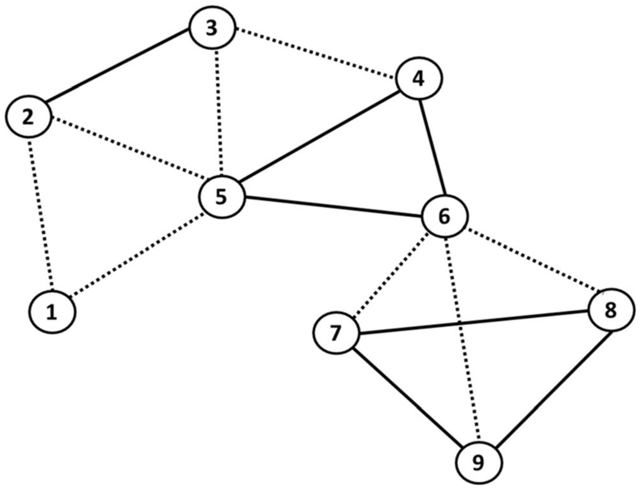


Fig. 1 Network *G* with 9 vertices

Two of these applications are described below. The work of Allignol et al. [1], addresses the problem of aviation authorities assigning flight levels (FLs) to different flights in a given planning period. Flight levels are the heights at which aircraft fly after take-off and before landing (known as "cruising altitudes"). Flight levels have a margin of 1000 ft (the difference in altitude between two consecutive levels). The problem is to assign different FLs to "incompatible" flights, i.e., flights whose trajectories may be less than 10 nautical miles apart at or near the same point in time. Assigning different FLs to incompatible flights involves establishing safety margins and avoiding accidents. The problem is stated as a graph in which each flight corresponds to a vertex and each edge joins two vertices (flights) that are compatible and can therefore fly without any type of conflict between them at the same FL. The idea is to divide the flights into groups such that all flights in a group are compatible. Therefore, each of these groups corresponds to a clique in the network. Each group of compatible flights (clique) is assigned a different FL. The objective is to minimize the number of groups and therefore the number of FLs to be used, or at least to ensure that this number does not exceed a previously established maximum number of FLs. Thus, the problem can be considered a real application of the CPP. However, other aspects are also considered, such as the costs of flying at each FL for each flight. Therefore, airlines request an ideal FL for each of their flights (normally the FL such that the flight would have the lowest fuel cost), and the allocation of FLs to flights must take these requests into account (by minimizing the number of allocations that are different from the ideal FLs, balancing the number of different allocations between airlines, etc.).

The work of Casado et al. [4] addresses the improvement of steel coil production by grouping processes. In the production of steel coils, raw materials (mainly iron and carbon but also manganese, silicon and niobium, among others) are first extracted. These ores are combined to produce steel in the form of slabs. A rolling mill then converts the slabs into coils. The characteristics of the coil determine the characteristics of the slabs. The problem concerns the production of a series of orders of steel coils requested by different customers. The coils in each order are the same as each other and different from those in other orders. Traditionally, it was ensured that there would be a one-to-one correspondence between the different orders of coils and the types of slabs being produced. However, the possibility of grouping the orders into compatible groups is now considered so that orders in the same group can be manufactured from the same type of slab. As an example, two orders of coils are compatible with respect to the type of steel if the types of steel required are of the same family (steels are divided into families, and within each family, they are ordered from lower quality to higher quality). In this case, the coils could be produced from slabs of the higher-quality steel type between the two initially needed. To check the compatibility between orders, other aspects, such as coil weights and widths, must be considered in addition to the type of steel. Grouping reduces the number of slab types that have to be manufactured: 1 type is needed per group or "cluster" instead of 1 per order. Thus, grouping has advantages such as greater continuity in the production process, cost reduction (in industrial processes with large equipment, the cost of stopping and restarting is very high), less chance of accidents and breakdowns, and improvement in storage and inventory management. This problem can be modeled as a graph in which the nodes are the orders and the edges link compatible nodes (orders). Cliques represent groups of compatible orders (and can therefore be produced by identical slabs). The objective is to minimize the number of groups and thus the number of types of slabs to be produced. It is also necessary to consider the costs that the grouping incurs (for example, by considering the cost of producing coils with steel of the same family but of a higher quality than that initially required). In this way, between two solutions with the same number of cliques, we can choose the one with the lowest cost.

The literature regarding theoretical and methodological aspects related to the CPP is less abundant than that related to other clique problems (especially the MCP). For example, Bhasker and Samad [2] formulated a new upper bound for the number of cliques, and they demonstrated that there is an optimal partitioning that includes a maximal clique (although not necessarily a maximum clique). The more recent publication of Sundar and Singh [19] developed two metaheuristic techniques based on evolutionary computation to solve the CPP. The proposed approaches were tested on 37 publicly available DI-MACS graph instances. More recently, Casado et al. [4] designed a method based on tabu search for an extension of the CPP to the field of manufacturing. This method was adapted to the CPP and tested on the same 37 DI-MACS instances.

The main contribution of this work is the design of a new heuristic method for the CPP. This method combines multistart strategies with tabu search. The most interesting characteristic of our method is the type of solution exploration that is used in the tabu search procedure. As described in more detail in the next section, the method allows infeasible solutions to be visited, which will in turn make the search more flexible. In this way, the method will be able to reach feasible solutions with fewer cliques and shorter computation times than previous methods, as shown in Section 3.

The rest of the article is organized as follows: Section 2 shows a mathematical formulation of the CPP, Section 3 describes the resolution method in detail, Section 4 presents different computational tests and Section 5 details the conclusions.

2 Mathematical formulation

Let $G = (V, E)$ be a graph with n vertices (i.e., $V = \{1, 2, \ldots, n\}$). Let k_{max} be an upper bound of the optimum number of cliques, and let C_k, $k = 1\ldots k_{max}$ denote each of the cliques. The problem can be formulated as a mathematical program with the following three sets of variables:

x_{ik}	A binary variable that equals one if product i belongs to clique C_k.
y_k	A binary variable that equals one if clique C_k is not empty
n_k	The number of vertices assigned to clique C_k

The problem can be formulated as follows:

$$\text{Minimize } \sum_{k=1}^{k_{max}} y_k \tag{1}$$

Subject to

$$x_{ik} \le y_k \qquad \forall i \in V, \forall k = 1\ldots k_{max} \tag{2}$$

$$\sum_{k=1}^{k_{max}} x_{ik} = 1 \qquad \forall i \in V \tag{3}$$

$$\sum_{i=1}^{n} x_{ik} \le n_k \qquad \forall k = 1\ldots k_{max} \tag{4}$$

$$\sum_{(i,j)\in E} x_{jk} - (x_{ik}-1)n \ge n_k - 1 \qquad \forall i \in V, \forall k = 1\ldots k_{max} \tag{5}$$

$$n_k \ge 0 \qquad \forall k = 1\ldots k_{max} \tag{6}$$

$$x_{ik} = \{0,1\} \text{ and } y_k = \{0,1\} \forall i \in V, \qquad \forall k = 1\ldots k_{max} \tag{7}$$

In this formulation, each variable y_k indicates whether clique k has an element or is empty; each variable x_{ik} indicates whether vertex i is assigned to cluster k; and each variable n_k indicates the number of elements of clique k. Target function (1) indicates the number of nonempty cliques. Restrictions (2) force $y_k = 1$ if any vertex i is assigned to clique k, restrictions (3) force the assignment of each vertex to a clique, restrictions (4) are cardinality restrictions on each clique, and restrictions (5) force each clique to be constituted by adjacent vertices. Last, an obvious value for k_{max} would be to take $k_{max} = n$. However, to avoid excessively long formulations of the instances, the value obtained by the *Constructive* procedure (described in Pseudocode 2 with $\alpha = 1$) was used as the value of k_{max}.

3 Resolution method

Let S represent a generic solution of the corresponding CPP, K be the number of nonempty cliques that constitute it, and C_1, $C_2\ldots C_K$ represent these cliques (in some cases, as an abbreviation, $S = \{C_1, C_2\ldots C_K\}$). Therefore, K is the target function to minimize.

The proposed method is a multistart procedure. Thus, it is an iterative process in which a solution is created in each iteration, which is then improved by a tabu search (TS) procedure. The novelty of this work is this TS procedure, which allows unfeasible solutions to be visited, thereby making the search more flexible. As described in Section 4, this strategy will allow the results obtained by previous procedures for this problem to be improved considerably. This TS procedure is described in detail below, as well as the entire proposed multistart method.

Specifically, subsection 3.1 explains the basic idea of the tabu search procedure and, more specifically, the moves it uses. Subsection 3.2 shows the flowchart of the tabu search and how moves are chosen. In subsection 3.3, some aspects of the tabu search procedure are explained in more detail and described in pseudocode. In subsection 3.4, the multistart procedure into which the tabu search procedure is integrated is described. Finally, in subsection 3.5, the differences from other heuristic methods for the CPP are explained.

3.1 Basic idea of the tabu search method: Movements

We begin with an initial feasible solution S, constituted by K nonempty cliques C_1, $C_2\ldots C_K$. K groups Gr_1, $Gr_2\ldots Gr_K$ are formed, which are initially identified with each clique; i.e., $Gr_k = C_k$, $k = 1, \ldots, K$. A group is "removed" or "deactivated", and its components are reassigned to other groups. This can lead to an unfeasible solution, since some of the groups may not be cliques; that is, there may be groups with unlinked pairs of vertices. Hereafter, for simplicity, each pair of unlinked vertices in the same group will be called an "incompatibility".

In the following steps, modifications or movements are made in the solution with the aim of reducing the number of incompatibilities. If the number of incompatibilities is reduced to zero, a feasible solution will be obtained (all groups are cliques) with one fewer clique ($K - 1$). The movement considered in each step is to change a vertex of one group to another group (considering only active groups).

To illustrate this process, an example is shown. Figure 1 shows an undirected network G with 9 vertices, and Fig. 2 presents an initial feasible solution, constituted by cliques $C_1 = \{1\}$, $C_2 = \{2, 3\}$, $C_3 = \{4, 5, 6\}$ and $C_4 = \{7, 8, 9\}$. Next, we define the initial groups $Gr_k = C_k$, $k = 1, \ldots, 4$.

In the next step (Fig. 3), group Gr_1 is removed ("deactivated"), and its elements are reassigned among the other groups. Specifically, vertex 1 is reassigned to group Gr_2. As shown in Fig. 4, there is an incompatibility in this group corresponding to vertices 1 and 3 (marked in red). Then, the movement that most reduces (or least increases) the number of incompatibilities is searched for and executed. Specifically, vertex 3 is moved to group Gr_3. This eliminates the incompatibility between vertices 1 and 3, but a new incompatibility appears between vertices 3 and 6 in group Gr_3 (Fig. 5). Last, vertex 6 is moved to group Gr_4, which removes all incompatibilities (Fig. 6). Thus, a new feasible solution is obtained, constituted by three cliques: {1, 2}, {3, 4, 5} and {6, 7, 8, 9}.

With this new feasible solution, this sequence or block of steps is repeated (removing a group, reassigning its elements, and performing movements to eliminate incompatibilities). Therefore, this is an iterative process in which the number of cliques is reduced by one unit in each block of steps. The entire process ends when it is not possible to remove the incompatibilities in one block.

3.2 Flowchart of the tabu search method

As mentioned above, this method consists of executing a sequence of blocks, where in each block, the number of clicks is reduced by one unit. The process ends when it is not possible

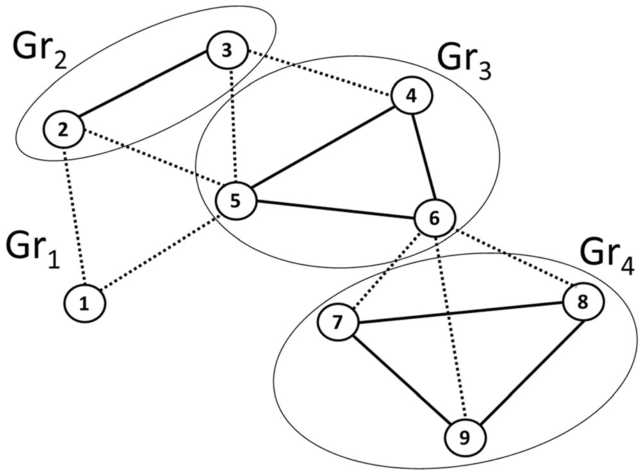


Fig. 3 Group Gr_1 is removed

to reduce the number of cliques in a block. The process for each block is to choose a group and "empty" it, i.e., assign the elements to other groups. This can lead to incompatibilities. In the next steps, moves are made to reduce these incompatibilities until they are eliminated, if possible. These moves consist of changing a vertex from one group to another. A block ends when a stop criterion is reached. This criterion is reached when all incompatibilities have been eliminated or when a certain number of iterations have elapsed without reducing the incompatibilities. If all incompatibilities have been eliminated, a feasible solution is found with one fewer clique (all groups are cliques), and a new block is started; otherwise, the process ends with the output being the best feasible solution found. Figure 7 shows the flowchart of this process.

In the flowchart of Fig. 7, C_k, $k = 1, \ldots, K$, are the cliques that define the initial solution; C_k^*, $k = 1, \ldots, K$, are the cliques

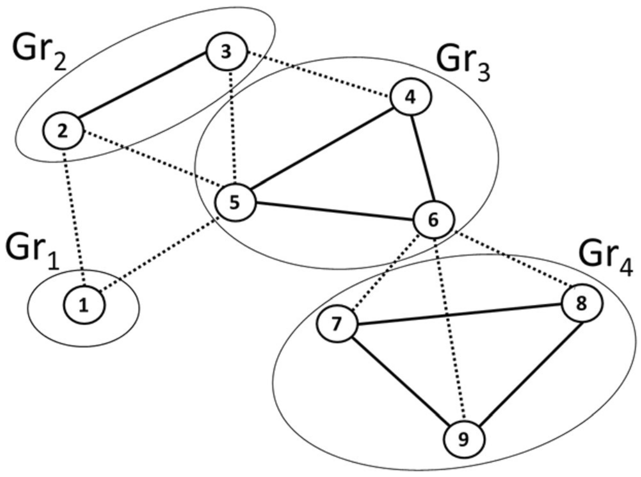


Fig. 2 Initial solution with 4 cliques

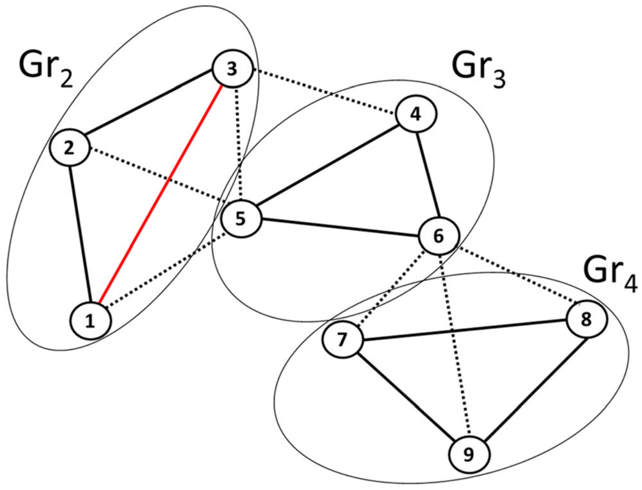


Fig. 4 Vertex 1 is reassigned to group Gr_2

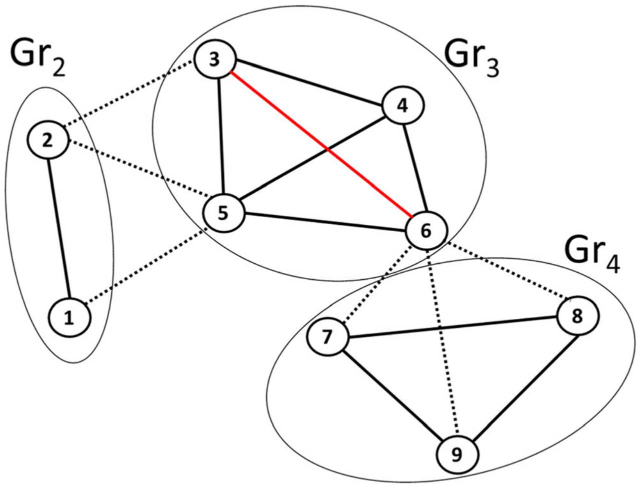


Fig. 5 Vertex 3 is moved to group Gr_3

in which the best solution is found and therefore in which the output of the method is stored. Note that some of these cliques can be empty. In this case, it is necessary to eliminate them and renumber the rest. *NInc* is the variable in which the number of incompatibilities is stored. The set of criteria or method by which to select a move in each step follows a basic tabu search strategy [7]. Initially, the best move is chosen, i.e., the one that reduces *NInc* the most or increases it the least; additionally, to prevent the algorithm from cycling, some moves are declared "tabu" and are not considered. In this case, returning a vertex to a group from which it was moved in recent iterations is declared tabu. The tabu status can be ignored if it results in a lower *NInc* value than that found in previous iterations of the block. The next subsection will

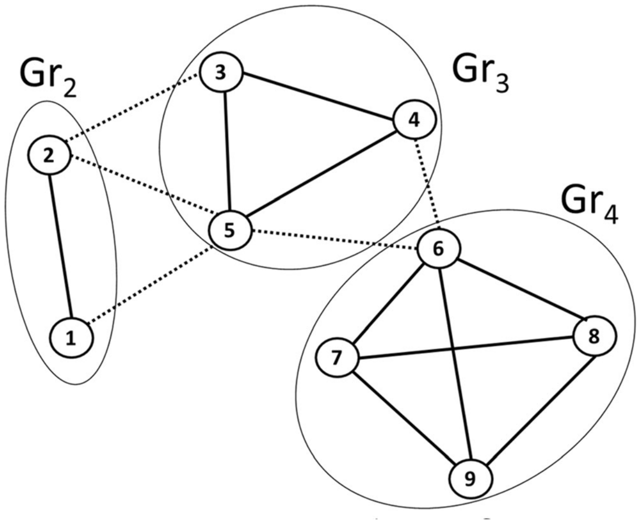


Fig. 6 Vertex 6 is moved to group Gr_4

explain the details of the tabu search method (which we name the *SteppedTabu* procedure) and describe its full pseudocode.

3.3 The *SteppedTabu* procedure: Details and pseudocode

Next, some aspects of the previous process are explained in more detail. The first aspect to explain is which criterion is used to select the group that is initially removed and the groups to which its vertices are reassigned. The idea is to select the group and perform the corresponding reassignments so that the number of "incompatibilities" increases as little as possible.

Let S be a solution (feasible or not) constituted by K groups, $Gr_1, Gr_2 \ldots Gr_K$; we define:

- *Active* = set of active groups
- $Grp(i)$ = index of the group to which vertex i belongs (i.e., $i \in Gr_{Grp(i)}$), $\forall i \in V$
- $Inc(i, k)$ = number of vertices of group Gr_k to which vertex i is not linked; more formally, $Inc(i, k) = |\{j \in Gr_k : j \neq i, (i, j) \notin E\}|, \forall i \in V, \forall k = 1..K$;

Similarly, we define:

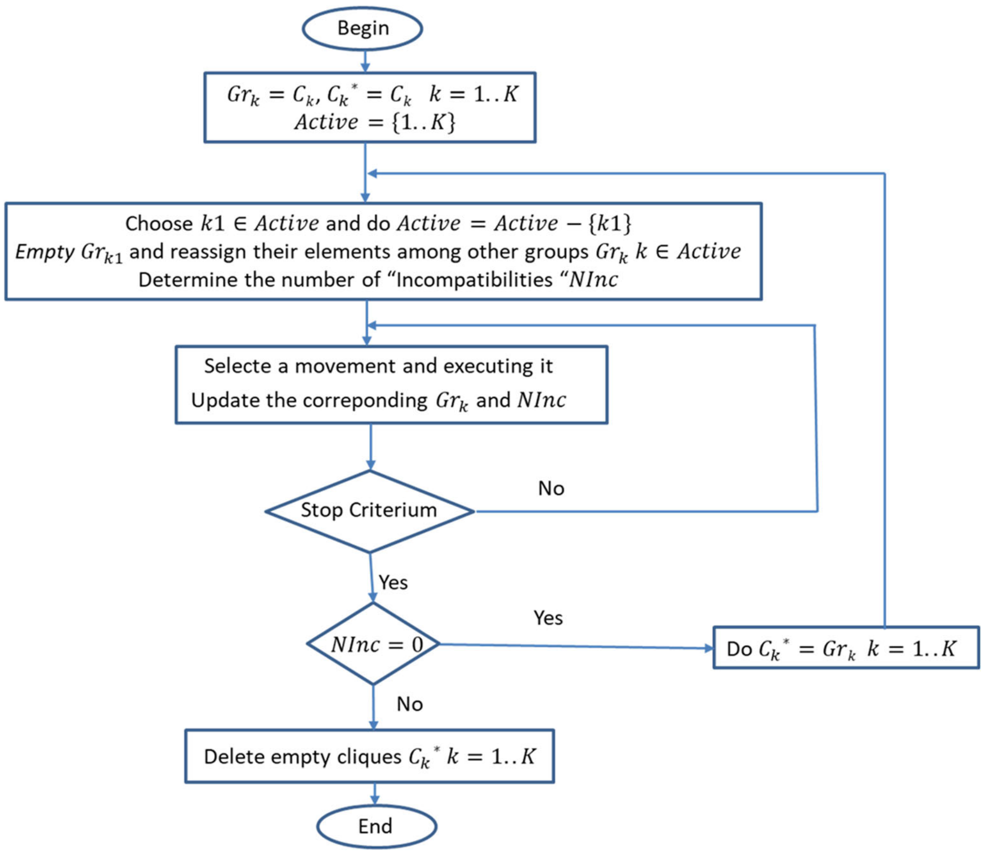
- $km(i) = argmin\{Inc(i, k) : Gr_k \in Active, \ \ i \notin Gr_k\}, \forall i \in V$
- -$SInc(k) = \sum_{i \in Gr_k} Inc(i, km(i))$
- -$k1 = argmin\{SInc(k) : k = 1..K\}$

Therefore, $km(i)$ indicates, for each vertex $i \in V$, the index of the "active" group (without considering the group it belongs to) with fewer vertices that are "incompatible" with i (i.e., vertices j such that $(i, j) \notin E$). Thus, it is the index of the group that would produce the smallest number of incompatibilities if i were "reassigned" to that group. $SInc(k)$ indicates the number of incompatibilities if group Gr_k is removed and each of its members i is reassigned to group $Gr_{kmin(i)}$. Last, $k1$ is the index of the group that would produce the smallest number of incompatibilities if it were removed and each of its elements i were reassigned to the corresponding groups $Gr_{km(i)}$. Consequently, group Gr_{k1} is eliminated.

As an example, the following table shows the initial values of $Inc(i, j)$, which correspond to the initial solution of Fig. 2. The lines correspond to the vertices, and the columns correspond to the groups. For each vertex i, the group that corresponds to $km(i)$ is highlighted in gray (Table 1).

Table 2 shows the values of *SInc*.

Therefore, the "best" group to be removed ($j1$) could be group.



Begin
$Gr_k = C_k, C_k^* = C_k \quad k = 1..K$
$Active = \{1..K\}$
Choose $k1 \in Active$ and do $Active = Active - \{k1\}$
Empty Gr_{k1} and reassign their elements among other groups Gr_k $k \in Active$
Determine the number of "Incompatibilities "$NInc$
Selecte a movement and executing it
Update the correponding Gr_k and $NInc$
Stop Criterium
No
Yes
$NInc = 0$
Yes
Do $C_k^* = Gr_k \quad k = 1..K$
No
Delete empty cliques C_k^* $k = 1..K$
End

Fig. 7 Flowchart of the tabu search method

Gr_1, Gr_2 or Gr_3. The draws are resolved in lexicographic order, although other criteria can be considered. Thus, in this case, we set $k1 = Gr_1$; we "deactivate" this group and reassign its only element, vertex 1, to group Gr_2. In the subsequent steps, group Gr_1 is no longer considered in the possible movements.

Next, for the following steps, we search for the "best" movement of a vertex i to a different group – that is, the movement that most reduces the number of incompatibilities in the solution. Specifically, $\forall i \in V, \forall\ k = 1..K\ i \notin Gr_k$. This increase *dif* (if it is positive) or reduction (if it is negative) is calculated as:

$$dif = Inc(i,k) - Inc(i, Grp(i)) \tag{8}$$

(This is the increase in incompatibilities in group k minus the decrease in incompatibilities in group $Grp(i)$ if the movement were to be executed.) Consequently, we continue to make use of auxiliary variables $Inc(i, j)$, which are conveniently updated. Table 3 shows the values of *Inc* and *dif* for the solution of Fig. 4.

The lowest value of *dif* is $dif = 0$, and the corresponding movement is to move vertex 3 to group Gr_3 (in this case, we use the lexicographic order of the draws). We execute this movement, obtaining the solution shown in Fig. 5. For the next step, Table 4 presents the values of *Inc* and *dif* for the solution of Fig. 5.

Table 1 Values of $Inc(i, j)$ for the solution of Fig. 2

Vertex	*Inc*			
	Gr_1	Gr_2	Gr_3	Gr_4
1	0	1	2	3
2	0	0	2	3
3	1	0	1	3
4	1	1	0	3
5	0	0	0	3
6	1	2	0	0
7	1	2	2	0
8	1	2	2	0
9	1	2	2	0

Table 2 Values of *SInc* for the solution of Fig. 2

Group→	Gr_1	Gr_2	Gr_3	Gr_4
SInc	1	1	1	3

Table 3 Values of *Inc* and *dif* for the solution of Fig. 4

Vertex↓*Group*→	*Inc*			*dif*		
	Gr_2	Gr_3	Gr_4	Gr_2	Gr_3	Gr_4
1	1	2	3		1	2
2	0	2	3		2	3
3	1	1	3		0	2
4	2	0	3	2		3
5	0	0	3	0		3
6	3	0	0	3		0
7	3	2	0	3	2	
8	3	2	0	3	2	
9	3	2	0	3	2	

The lowest value of *dif* (−1) corresponds to moving vertex 6 to group Gr_4. The execution of this movement removes all incompatibilities, resulting in a feasible solution (Fig. 6).

As observed in the previous example, in some cases, it is convenient to admit movements that do not improve the solution (i.e., movements that do not reduce the number of incompatibilities) to prevent the blockage and termination of the process. Thus, as shown in Table 3, in the solution presented in Fig. 4, it is not possible to find any movement that reduces the number of incompatibilities. However, we do not end the process at this point, and we perform the best possible move (moving vertex 3 to groupGr_3), although it does not improve the solution. Then, vertex 6 of group Gr_3 is moved to Gr_4, obtaining a solution without incompatibilities. On the other hand, if we admit movements that do not improve the solution (and even movements that may worsen it), the process must be equipped with mechanisms that prevent it from cycling. Specifically, if a vertex has just left a group, the aim is to prevent it from returning to the group in subsequent iterations. To this end, we define *tabumatrix*(i, k) $\forall\ i \in V$, $\forall\ k = 1..K$, as follows:

tabumatrix(i, k)	Number of iterations or steps (within each block) in which vertex i exits group Gr_k

Table 4 Values of *Inc* and *dif* for the solution of Fig. 5

Vertex	*Inc*			*dif*		
	Gr_2	Gr_3	Gr_4	Gr_2	Gr_3	Gr_4
1	0	3	3		3	3
2	0	2	3		2	3
3	1	1	3	0		2
4	2	0	3	2		3
5	0	0	3	0		3
6	2	1	0	1		−1
7	2	3	0	2	3	
8	2	3	0	2	3	
9	2	3	0	2	3	

Therefore, the movement defined by vertex i and group Gr_k is declared "tabu" (and its execution is prevented) if:

$$iter \leq tabumatrix(i, k) + tenure, \tag{9}$$

where *iter* is the number of the current iteration within each block. The parameter t*enure* indicates the number of iterations in which the return of i to group Gr_k is declared tabu after the moment at which it leaves it. Excessively high values of *tenure* may lead the algorithm to be greatly hindered, and very low values can make it cycle. Consequently, the selection of this parameter is critical for the satisfactory functioning of the process. On the other hand, the tabu state of a movement can be ignored if this movement produces a solution with fewer incompatibilities than any solution visited during that block ("aspiration criterion").

In summary, the entire process is a sequence of blocks. Each block searches for a solution with one fewer clique than the previous block. Therefore, this is a descending "stepped" process (each block is a "step" with one fewer clique than the previous block). Each block follows a tabu search strategy to obtain a feasible solution (without incompatibilities). We call this process or algorithm *TabuStepped*, and it is thoroughly described in Pseudocode 1. In this pseudocode, the main variables are:

- S: a feasible initial solution, where K is the number of cliques that constitute it. These cliques are represented as $C_1, C_2 \ldots C_K$.
- S^*: the final solution, that is, the best feasible solution found, where K^* is the number of cliques that constitute it (represented as $C_1^*, C_2^* \ldots C_{K^*}^*$).
- Gr_k, $k = 1, \ldots, K$: the groups that make up the solutions (feasible or not) visited during the process.
- *Kf*: number of active groups at each moment.

Pseudocode 1 *SteppedTabu* **Procedure**

```
Procedure $SteppedTabu(tenure, maxiter, S, K;$ output: $S^*, K^*)$

1.  Let $C_1, C_2 \ldots C_K$ do $Gr_k = C_k, k = 1,..,K$ (initial groups)
2.  Do $Inc(i,k) = |\{j \in Gr_k : j \neq i, (i,j) \notin E\}|\ \ \forall i \in V, \forall k = 1..K$
3.  Do $Active\ = \{1,2, \ldots K\}$ and $Kf = K$
4.  Do $S^* = S\ (C_k^* = C_k\ \forall k = 1,..,K)$ and $K^* = K$
Repeat
    5.  $km(i) = argmin\{Inc(i,k): k \in Active, i \notin Gr_k\}, \forall i \in V$
    6.  $SInc(k) = \sum_{i \in G_k} Inc(i, km(i)), \forall k \in Active$
    7.  $k1 = argmin\{SInc(k): \in Active\}$
    8.  $\forall i \in Gr_{k1}$:
        8a. $Gr_{km(i)} = Gr_{km(i)} + \{i\}$ (and $Grp(i) = km(i)$)
        8b. $\forall j \in V, (i,j) \notin E$:    $Inc(j,k1) = Inc(j,k1) - 1$
                                   $Inc(j, km(i)) = Inc(j, km(i)) + 1$
    9.  $Gr_{k1} = \emptyset, Active = Active - \{k1\}, Kf = Kf - 1$
    10. $NInc = SInc(k1)$
    11. $NIncbest = NInc, iter = 0, iterbest = iter$
    12. $tabumatrix(i,k) = -tenure\ \forall i \in V, \forall k \in Active$
    While $(NInc > 0)$ and $(iter \leq iterbest + maxiter)$ do
        begin
        13. $iter = iter + 1$
        14. $difb = \infty$
        15. $\forall i \in V, \forall k \in Active,\ i \notin Gr_k$ do:
            15a.  $dif = Inc(i,k) - Inc(i, Grp(i))$
            15b.  If $((NInc + dif < NIncbest)$ or $(iter > tabumatrix(i,k) + tenure))$
                      and $(dif < difb)$ then $difb = dif,\ ib = i$ and$kb = k$
        16. If $difb < \infty$ then:
            16a.  $kout = Grp(ib)$
            16b.  $Gr_{kb} = Gr_{kb} + \{ib\}, Gr_{kout} = Gr_{kout} - \{ib\}\ \ (Grp(ib) = kb)$
            16c.  $NInc = NInc + difb$
            16d.  $\forall j \in V, (ib,j) \notin E$:  $Inc(j, kout) = Inc(i, kout) - 1$
                                       $Inc(j, kb) = Inc(i, kb) + 1$
            16e.  $tabumatrix(ib, kb) = iter$
            16f.  If $NInc < NIncbest$ then $NIncbest = NInc$ and $iterbest = iter$
        end
    17. If $NInc = 0$ then save $S^*$ (i.e., $C_k^* = Gr_k\ \forall k = 1,..,K$) and $K^* = Kf$
until $NInc > 0$
18. Remove from $S^*$ the empty cliques and renumber those that are not empty from 1 to $K^*$
```

In Pseudocode 1, steps 1–4 represent the beginning of the entire process (initiate *S**, *Gr*, *Inc*, *Kf*). The repeat-until loop (steps 5–17) represents the execution of each block. Thus, steps 5–9 determine which group *k*1 is to be deactivated, reassign its elements to other groups and determine the number of incompatibilities (*NInc*). Steps 10–12 initiate the variables that will be used in the *NInc* reduction phase (*iter*, *iterbest*, *tabumatrix*, *Niterbest*). The do-while loop (steps 13–16) represents each of the iterations of this phase. Thus, step 15 explores the entire set of movements, and step 16 executes the best movement and updates the different variables. Once this do-while loop ends, step 17 verifies whether the number of incompatibilities was reduced to 0. If this is the case, the obtained solution is saved in S^*, and a new block is executed. Otherwise, the algorithm ends.

The auxiliary variable *NIncbest* indicates the minimum number of incompatibilities during the *NInc* reduction phase; *NIncbest* is used in the "aspiration criterion" (step 15b, which verifies whether (*NInc* + *dif* < *NIncbest*)). The variable *iterbest* indicates the iteration in which *NIncbest* was modified (step 16b). *maxiter* is the parameter of the algorithm that indicates the maximum number of iterations of the do-while loop after the modification of *NIncbest*. Therefore, *maxiter* and *NIncbest* determine the stopping criterion for this loop.

In the search for the best movement (step 15), a movement is considered if it improves *Nincbest* (*NInc* + *dif* < *NIncbest*) or if it is not tabu (*iter* > *tabumatrix*(*i*, *k*) + *tenure*). The values of the best movement found are saved in the variables *difb* (variation in the number of incompatibilities), *ib* (vertex) and *kb* (group).

3.4 *MultiStartStepped* (procedure)

This *SteppedTabu* procedure is inserted into the multistart procedure, as previously stated. The initial solutions that are subsequently improved by the *SteppedTabu* procedure are built with the constructive method proposed in Casado et al. [4], which is briefly described in Pseudocode 2.

Pseudocode 2 *Constructive* (procedure)

```
Constructive procedure ($\alpha$; output: $S, K$)
1.  $K = 0, U = V$
Repeat
    2.  $K = K + 1, C_K = \emptyset, U' = U$
    Repeat
        3.  $L_i = \{j \in U' : (i, j) \in E\}$ and $g(i) = |L_i|\ \ \forall i \in U'$
        4.  $g_{max} = max\{g(i) : i \in U'\}$ and $g_{min} = min\{g(i) : i \in U'\}$
        5.  $LC = \{i \in U' : g(i) \geq \alpha g_{max} + (1-)g_{min}\}$
        6.  Choose $i^* \in LC$ randomly
        5.  $C_K = C_K + \{i^*\}$
        6.  $U = U - \{i^*\}$
        7.  $U' = L_{i^*}$
    until $U' = \emptyset$
until $U = \emptyset$
8.  $S = \{C_1, C_2 \dots C_K\}$
```

The "guide" function $g(i)$ measures the suitability of each candidate vertex to be chosen. The parameter α takes values between 0 and 1 and regulates the degree of randomness of the method. If $\alpha = 1$, the process always produces the same solution (except in cases in which there is an iteration with more than one vertex $i \in U'$ corresponding to *gmax*). If $\alpha = 0$, the values of $g(i)$ are irrelevant, and the process is totally random. A more detailed explanation of this method is provided in Casado et al. [4]. The multistart procedure (which we name *MultiStartStepped* or MSS) is described in Pseudocode 3.

Pseudocode 3 *MultiStartStepped* (procedure)

```
Procedure $MultiStartStepped$
$iter = 0, iterbest = 0, K^{best} = \infty$
Repeat
    1. $iter = iter + 1$
    2. Execute $Constructive(\alpha, S, K)$
    3. Execute $SteppedTabu(tenure, maxiter, S, K, S^*, K^*)$
    4. if $(K^* < K^{best})$ then $S^{best} = S^*, K^{best} = K^*$ and $iterbest = iter$
until $(iter - iterbest \geq maxiterMSS)$
```

As can be observed, the final solution obtained is S^{best}, with the corresponding number of cliques K^{best}. Therefore, this procedure depends on the following parameters: α (from the constructive procedure), *tenure*, *maxiter* (from the *SteppedTabu* procedure) and *maxiterMSS*. The next section analyses the different computational tests used to measure the performance of this procedure.

3.5 Differences with other recent heuristics

To the best of our knowledge, there are 3 recent heuristic methods for the CPP: two methods proposed by Sundar and Singh [19] and one method proposed in Casado et al. [4]. The two methods proposed by Sundar and Singh [19] are evolutionary methods that work on a population of solutions that can interact with each other. These two evolutionary methods are based on the genetic algorithm and an artificial bee colony. In the case of the genetic algorithm, the Selection, Crossover, Repair, Mutation and Replacement operators/procedures are described in detail. In the case of the artificial bee colony method, the procedure for building the neighboring solutions, selecting solutions for an onlooker bee, etc., are described. In both methods, the same procedure is used to build the initial population, and only feasible solutions are considered.

Our MSS method, as explained above, is a multistart method in which at each iteration, a solution is constructed that is improved by a tabu search procedure. At each step, it operates on a single solution that continually changes. Therefore, there is no interaction between different solutions, and the above-mentioned operators are not used. It also allows infeasible solutions to be visited.

The method proposed in Casado et al. [4] has a similar strategy to ours: it is a multistart method that uses a tabu search procedure in the improvement phase. The difference between this tabu search procedure and the one proposed in our work is that the procedure of Casado et al. [4] does not allow moves to infeasible solutions. Neighboring solutions must be feasible; i.e., the sets must be cliques (no "incompatibilities" are allowed). To determine which solutions are better, two hierarchical criteria are used: the number of cliques and the "imbalance" in the number of elements in the cliques. As a measure of "imbalance", the sum of the squares of the number of elements in each clique is taken. Solutions with fewer cliques are preferred, and in the case of equality in the number of cliques, the solution with the highest "imbalance" is chosen.

Thus, among the 3 solutions

$$S_1 = \{\{1,2,3,4,5,6\},\{7,8,9,10\}\}$$
$$S_2 = \{\{1,2,3,4,5,6\},\{7,8\},\{9,10\}\}$$
$$S_3 = \{\{1,2,3,4\},\{5,6,7\},\{8,9,10\}\}$$

solution S_1 is preferred over S_2 and S_3 since S_1 consists of two cliques while S_2 and S_3 consist of 3. On the other hand, S_2 is preferred over S_3 since the "imbalance" of S_2 is 44 (36 + 4 + 4) and that of S_2 is 34 (16 + 9 + 9). The idea of using the imbalance criterion is to force empty clusters with fewer elements.

Our tabu search procedure is more aggressive: starting from a feasible solution, a clique is eliminated, and its elements are distributed among the other cliques, which produces "incompatibilities", i.e., infeasible solutions. In the next steps, the procedure searches for solutions with fewer incompatibilities until it reaches a solution without incompatibilities, that

Table 5 Instances for parameter fitting

Instance		
Name	*n*	*density*
C125.9	125	0.8985
C1000.9	1000	0.9011
C2000.5	2000	0.5002
brock200_2	200	0.4963
keller6	3361	0.8182
p_hat300–1	300	0.2438
p_hat700–1	700	0.2493
p_hat1500–1	1500	0.2534
DSJC500_5	500	0.5020

is, a feasible solution with one fewer clique than the previous feasible solution. This process is repeated with the new feasible solution. Allowing infeasible solutions to be visited gives more flexibility to the search, and as will be seen below, the resulting MSS method produces better solutions than the 3 methods above.

4 Computational tests

This section describes a series of computational tests used to compare the performance of our multistart algorithm. The algorithm was implemented in Delphi (Object Pascal) with the development environments Rad Studio 10.3 and 11. The tests with this code were conducted using a workstation with an AMD 3990X 2.9 GHz processor with 256 Gb RAM. We also used CPLEX 20.1 in our tests. This section is divided into the following parts: Subsection 4.1 describes the parameter fitting carried out in our method, Subsection 4.2 compares our method with the commercial software CPLEX, and Subsection 4.3 compares the results with other recent heuristic results for this problem.

4.1 Parameter fine-tuning

Parameter fitting was carried out using a set of 9 instances of the DIMACS library. Table 5 shows this set and its characteristics. Specifically, it indicates the names, sizes (n), and densities (percentages of links in the network over the total possible number of links) of the instances.

The choice of instances has been made in such a way as to combine instances of different size and density. Note that there are 3 instances of small size (*C125.9*, *brock200_2* and *p_hat300–1*), 3 instances of medium size (*DSJC500_5*, *p_hat700–1* and *C1000.9*) and 3 instances of large size (*p_hat1500–1*, *C2000.5* and *keller6*). On the other hand, there are 3 low density instances (*p_hat300–1*, *p_hat700–1* and *p_hat1500–1*), 3 medium density instances (*C2000.5*, *brock200_2* and *DSJC500_5*) and 3 high density instances (*C125.9*, *C1000.9* and *keller6*).

As previously stated, our MSS method has 4 parameters: α, *tenure*, *maxiter* and *maxiterMSS*. The parameter α indicates the degree of randomness of the constructive procedure, and the parameter *tenure* regulates the number of "tabu" movements in the *SteppedTabu* procedure. The parameters *maxiter* and *maxiterMSS* are used as stopping criteria in the tabu procedure and in the general MSS procedure. To fit the other two parameters (α and *tenure*), we fixed the values $maxiter = 10 \cdot n$ and $maxiterMSS = 20$. For α, the values considered were $\alpha = 0, 0.1, 0.5, 0.9, 0.99$ and 1. The values of *tenure* considered were $tenure = {}^{n}/_{2}, n, 2 \cdot n$ and $5 \cdot n$. Of all the combinations, the one that yielded the best results was $\alpha = 0.99$ and $tenure = {}^{n}/_{2}$. With these values, we analyzed the parameters *maxiter* and *maxiterMSS*. It was determined that there were no significant improvements with values above $maxiter = 10 \cdot n$ and $maxiterMSS = 100$. Therefore, these values were used in the rest of the tests. Finally, it should be noted that the stopping criterion used by the *MultiStartStepped* procedure is to achieve *maxiterMSS* consecutive starts without improvement in solution quality subject to a maximum of 3600 seconds.

Table 6 Results obtained by CPLEX and MSS

Instance		Cplex			MSS	
n	*density*	Lower Bound	*val*	C.T.	*val*	C.T.
10	0.2667	5	**5**	0.11	**5**	< 0.001
10	0.6000	4	**4**	0.08	**4**	< 0.001
10	0.6222	4	**4**	0.06	**4**	< 0.001
20	0.3211	8	**8**	0.53	**8**	< 0.001
20	0.4842	6	**6**	0.27	**6**	< 0.001
20	0.7579	4	**4**	0.17	**4**	< 0.001
30	0.3172	10	**10**	22.94	**10**	< 0.001
30	0.4621	8	**8**	1.53	**8**	< 0.001
30	0.7287	5	**5**	0.47	**5**	< 0.001
40	0.2910	9	**12**	> 3600	**12**	< 0.001
40	0.4872	9	**9**	39.53	**9**	< 0.001
40	0.7000	6	**6**	0.61	**6**	< 0.001
50	0.2873	13	**14**	> 3600	**14**	< 0.001
50	0.4743	8	**10**	> 3600	**10**	0.001
50	0.7086	7	**7**	7.56	**7**	0.001
60	0.3158	11	16	> 3600	**15**	0.051
60	0.5051	9	**11**	> 3600	**11**	0.001
60	0.7209	7	**7**	110.52	**7**	0.001
70	0.3072	9	19	> 3600	**17**	0.386
70	0.5064	7	14	> 3600	**12**	0.006
70	0.6894	8	**8**	804.00	**8**	0.002
80	0.3108	8	24	> 3600	**18**	0.121
80	0.5013	7	16	> 3600	**13**	0.003
80	0.7022	6	9	> 3600	**8**	0.331
90	0.3051	9	23	> 3600	**20**	0.213
90	0.4901	7	18	> 3600	**15**	0.002
90	0.6994	7	**9**	> 3600	**9**	0.007
100	0.2974	10	26	> 3600	**23**	0.006
100	0.4998	8	19	> 3600	**15**	0.036
100	0.6927	7	**10**	> 3600	**10**	0.002
110	0.3076	10	28	> 3600	**23**	0.821
110	0.4952	7	20	> 3600	**16**	2.079
110	0.7061	6	12	> 3600	**10**	0.200
120	0.2992	11	31	> 3600	**26**	0.509
120	0.5091	8	23	> 3600	**17**	0.244
120	0.6971	6	13	> 3600	**11**	0.045

Table 7 Results of SSGGA, GABC and MSST (10 runs)

Instance			SSGGA			GABC			MSS		
name	*n*	den.	Best	Avg.	*Time*	Best	Avg.	*Time*	Best	Avg.	*Time*
C125.9	125	0.898	**6**	6.1	*1.25*	**6**	6.0	*0.08*	**6**	6.0	*1.63*
C250.9	250	0.899	10	10.0	*2.93*	10	10.0	*0.5*	**9**	9.0	*2.54*
C500.9	500	0.900	16	16.5	*13.44*	15	15.7	*5.44*	**14**	14.0	*1.73*
C1000.9	1000	0.901	26	27.1	*63.52*	25	25.0	*59.95*	**23**	23.0	*10.55*
C2000.9	2000	0.900	45	45.9	*345.56*	42	42.0	*527.8*	**40**	40.0	*365.25*
C2000.5	2000	0.500	**173**	174.1	*710.09*	178	178.9	*1167.49*	178	178.8	*3228.18*
C4000.5	4000	0.500	**315**	315.8	*5227.38*	321	322.7	*8627.66*	325	326.3	*3600.00*
MANN_a27	378	0.990	**4**	4.0	*2.12*	**4**	4.0	*2.59*	**4**	4.0	*0.55*
MANN_a45	1035	0.996	**4**	4.0	*11.82*	**4**	4.0	*60.89*	**4**	4.0	*2.28*
MANN_a81	3321	0.999	**4**	4.0	*108.96*	**4**	4.0	*2726.52*	**4**	4.0	*56.04*
brock200_2	200	0.496	**25**	25.9	*2.47*	27	27.4	*1.29*	26	26.0	*11.73*
brock200_4	200	0.658	**18**	18.5	*2.52*	19	19.4	*1.04*	**18**	18.1	*3.45*
brock400_2	400	0.749	25	25.5	*10.99*	25	25.4	*7.54*	**24**	24.0	*74.09*
brock400_4	400	0.749	25	25.9	*11.44*	25	25.4	*7.07*	**23**	23.8	*16.83*
brock800_2	800	0.651	57	57.7	*51.59*	58	58.4	*72.11*	**56**	56.8	*141.12*
brock800_4	800	0.650	57	58.0	*46.59*	58	58.3	*74.51*	**56**	56.8	*140.00*
gen200_p0.9_44	200	0.900	9	9.0	*1.41*	8	8.6	*0.29*	**7**	7.3	*1.38*
gen200_p0.9_55	200	0.900	**7**	7.6	*1.63*	**7**	7.5	*0.23*	**7**	7.0	*1.28*
gen400_p0.9_55	400	0.900	14	14.2	*7.03*	14	14.0	*3.41*	**12**	12.3	*11.46*
gen400_p0.9_65	400	0.900	13	13.9	*6.88*	13	13.9	*2.74*	**11**	11.4	*12.90*
gen400_p0.9_75	400	0.900	13	13.8	*5.83*	13	13.1	*3.03*	**10**	10.7	*15.30*
hamming8–4	256	0.639	**16**	16.0	*7.03*	**16**	16.0	*1.18*	**16**	16.0	*3.86*
hamming10–4	1024	0.829	38	38.0	*62.89*	37	37.2	*107.48*	**34**	34.1	*89.12*
keller4	171	0.649	20	20.2	*2.14*	21	21.3	*0.82*	**19**	19.7	*30.38*
keller5	776	0.752	44	45.5	*39.98*	47	47.1	*68.85*	**42**	43.0	*1822.29*
keller6	3361	0.818	100	101.5	*1537.98*	103	103.1	*4459.84*	**91**	93.4	*3491.51*
p_hat300–1	300	0.244	66	66.5	*3.61*	70	70.8	*2.56*	**64**	65.1	*19.38*
p_hat300–2	300	0.489	45	45.6	*4.07*	45	46.4	*3.1*	**42**	42.4	*179.68*
p_hat300–3	300	0.744	22	22.5	*4.21*	22	22.7	*3.11*	**19**	19.7	*139.81*
p_hat700–1	700	0.249	135	137.4	*19.93*	146	147.4	*36.63*	**131**	132.9	*3600.00*
p_hat700–2	700	0.498	93	94.1	*19.29*	93	94.0	*35.44*	**85**	87.4	*1477.47*
p_hat700–3	700	0.748	44	44.3	*25.31*	43	44.2	*32.86*	**37**	38.1	*109.82*
p_hat1500–1	1500	0.253	271	271.8	*120.12*	284	285.4	*263.04*	**253**	256.1	*3474.96*
p_hat1500–2	1500	0.506	175	176.4	*103.60*	176	177.7	*254.52*	**157**	162.8	*3216.74*
p_hat1500–3	1500	0.754	80	81.2	*160.47*	79	79.9	*290.49*	**67**	69.0	*3481.54*
DSJC500_5	500	0.502	**54**	54.4	*14.72*	56	56.7	*20.16*	55	55.0	*543.81*
DSJC1000_5	1000	0.500	**96**	97.3	*76.08*	100	101.0	*143.92*	99	99.0	*1946.58*

4.2 Tests with CPLEX

Since the CPP is an NP-hard problem, it is reasonable to expect an excessive computation time to be needed to solve large instances exactly. However, it would be interesting to determine the evolution of these computation times as a function of the size of the instances and determine the maximum size of the instances that can be solved exactly within a reasonable time. If this size is small, it would justify the use of heuristic strategies, such as the one proposed in this work. It would also be interesting to know whether our strategy can find the optimum solution in instances in which such an optimal solution is known and, if so, determine the deviation from this optimal solution. In this section, the commercial software CPLEX is used to solve small instances exactly. The results obtained by CPLEX are compared with those obtained by our

MSS. Specifically, instances of sizes $n = 10, 20, 30, \ldots, 120$ were randomly generated. For each value of n, three values of different densities were considered: low (approximately 0.3), medium (approximately 0.5) and high (approximately 0.7). To avoid excessive computation times, the time used per instance and method was limited to 3600 seconds. Table 6 shows the obtained results. For each method, the value of the best solution found (*val*) is indicated, and the computation time is shown in seconds (C.T.). In the case of CPLEX, the value of the lower bound obtained is added. If this lower bound coincides with *val*, then the solution obtained by CPLEX is optimal, and the process ends (if the process did not end within 3600 seconds, this is indicated by "> 3600" in the C.T. column). For each instance, the value of the best solution obtained is indicated in bold.

In Table 6, the following can be observed:

– CPLEX was able to finish, thereby obtaining the optimal solution, for instances up to size $n = 30$. In the instances of sizes $n = 40$, 50, 60 and 70, CPLEX was able to finish in some cases (those of greater density) and not in others. After $n = 80$, CPLEX was not able to finish in any instance.
– It seems that CPLEX behaves better for instances of greater density than those of lower density: among the small instances ($n \leq 30$), the computation time was shorter for instances of greater density; among the medium-sized instances ($40 \leq n \leq 70$), CPLEX was able to finish for instances of greater density; and among the large instances ($n \geq 80$), the gap between the lower bound and the value obtained was smaller for the instances of greater density.
– MSS was able to obtain the best result in all instances: in 20 instances, MSS and CPLEX obtained solutions with the same value, and in 16 instances (in most of the larger instances), MSS obtained strictly better solutions. Moreover, the computation time was very short (it exceeded one second in only one instance). Therefore, in the instances in which the optimal solution was known (those where CPLEX finished), MSS reached the optimal solution with very short calculation times.

In summary, only small instances were resolved exactly ($n \leq 30$, and in some instances of $40 \leq n \leq 70$). Larger instances seem to require the use of heuristic techniques, such as the MSS method proposed. In all the instances in which the optimal solution is known, our MSS method reached the optimal solution.

4.3 Computational tests against recent heuristics

As shown in the previous subsection, this problem can only be solved exactly in small instances. This justifies the development of heuristic methods such as our MSS method. In this subsection, we compare our method with other recent heuristics in the literature for this problem. Specifically, we consider the two methods proposed in Sundar and Singh [19]: the method based on genetic algorithms (SSGGA) and the method based on the artificial bee colony (GABC). To check the performance of these methods, the authors used a set of 37 instances from the well-known DIMACS library (http://dimacs.rutgers.edu/programs/challenge/). These instances were selected due to their difficulty (the presence of overlapping cliques). Then, Casado et al. [4] designed a method based on tabu search (MSTS) for an extension of the CPP to a manufacturing problem. This method was adapted to the CPP and tested in these 37 DIMACS instances. To compare our MSS method with these heuristics, we executed it in these same instances.

In the first set of tests, MSS was run 10 times (as were SSGGA and GABC), and the results obtained are shown in Table 7 together with those of SSGGA and GABC reported in Sundar and Singh [19]. Specifically, Table 7 shows the name, size (n) and density ("den.") of each instance; for each method and instance, the best result of the 10 runs ("Best"), the average of the results of the 10 runs ("Avg.") and the average execution time over the 10 runs ("Time") are shown. For each instance, the best solution is marked in bold.

Regarding the best values obtained ("Best" columns) by each method, our MSS method finds the best solution in 32 out of 37 instances, the SSGGA method finds the best solution in 11 instances, and the GABC method finds the best solution in 6 instances. On the other hand, the MSS method is strictly better than SSGGA in 25 out of 37 instances and strictly worse in 5. The MSS method is strictly better than GABC in 29 out of 37 instances and strictly worse in 1. Regarding the mean values ("Avg." columns), our MSS finds the best mean value in 32 out of 37 instances, the SSGGA method finds the best

Table 8 Results of the Wilcoxon signed rank tests

		n^*	W^+	W^-	$minW$	p-tail	Z score	p-tail z
Best	SSGGA-MSS	30	390.5	74.5	74.5	< 0.001	12.8598	< 0.00001
	GABC-MSS	30	444	21	21	< 0.001	8.6248	< 0.00001
Avg.	SSGGA-MSS	33	486.5	74.5	74.5	< 0.001	3.8925	0.00005
	GABC-MSS	32	506	22	22	< 0.001	4.8531	< 0.00001

mean value in 9 instances, and the GABC method finds the best mean value in 5 instances. On the other hand, the MSS method obtains strictly better mean values than SSGGA in 28 out of 37 instances and strictly worse in 5. The MSS method obtains strictly better mean values than GABC in 31 out of 37 instances and strictly worse in 1.

To reinforce the conclusions that can be drawn from Table 7, the following Wilcoxon signed rank tests with the corresponding null and alternative hypotheses are proposed:

$$-H_0 : \mu_{SSGGA} \leq \mu_{MSS}; H_1 : \mu_{SSGGA} > \mu_{MSS} \quad (10)$$

$$-H_0 : \mu_{GABC} \leq \mu_{MSS}; H_1 : \mu_{GABC} > \mu_{MSS} \quad (11)$$

These two tests are performed both for the best values ("Best" columns in Table 7) and for the average values ("Avg." columns). Therefore, 4 tests are performed. The results of these tests are shown in Table 8. Each table of this test shows the number of instances in which there is no tie (n^*); the sum of ranks with a positive difference (i.e., where the values of the solutions of SSGGA and GABC are higher than those of MSST), denoted by W^+; the sum of ranks with a negative difference (W^-); the lowest values of W^+ and W^- ($minW$) and the corresponding probability tail (p-tail); and the z value obtained (Z score) and the corresponding probability tail (p-tail z).

As seen in Table 8, considering both the *minW* values and the Z score values, the corresponding probability tails are insignificant: the null hypothesis is rejected in all 4 tests, and therefore, it can be concluded that MSST obtains significantly better values than SSGGA and GABC (both in terms of the best solution found and in terms of mean values).

In a second set of tests, our MSS is run once, and its results are shown in Table 9 together with those obtained from MSTS (reported in Casado et al. [4]). Table 9 shows the data for each instance (as in Table 7), and for each method and instance, the value of the solution obtained ("Val") and the elapsed computation time to obtain that solution ("Time-Best") are shown. The best solution of each instance is marked in bold.

As seen in Table 9, in the comparison with MSTS, our MSS obtains the best solution in all instances (37), while MSTS obtains the best solution in 7. Therefore, MSS strictly outperforms MSTS in 30 instances, while in the other 7 instances, both methods obtain solutions of equal quality. Applying the Wilcoxon signed rank test, there are 30 instances with strictly positive differences and no negative differences. Therefore, following the notation of Table 8, n^*= 30, W^+= 465, W^-= 0 and $minW$=0. Therefore, our MSS is significantly better than MSTS.

Therefore, our MSS method considerably improves the results of the previous methods. Regarding the computation times, in some instances where MSS exceeds the other methods, the computation time of MSS is much longer than those of the other methods. This happens in instances *keller4*, *keller5*, *p_hat300–2*, *p_hat300–3*, *p_hat700–1*, *p_hat700–2*, *p_hat700–3*, *p_hat1500–1*, *p_hat1500–2* and *p_hat1500–3*. However, observing the evolution of the results with respect to the computation times in these instances, it can be concluded that our method achieves better results than the previous methods in a similar or shorter computation time.

Table 9 Results of MSTS and MSS (1 run)

Instance			MSTS		MSS	
Name	*n*	den.	Val.	Time-Best	Val.	Time-Best
C125.9	125	0.898	**6**	0.063	**6**	0.141
C250.9	250	0.899	**9**	1.714	**9**	0.016
C500.9	500	0.900	15	1.605	**14**	0.188
C1000.9	1000	0.901	24	110.032	**23**	20.589
C2000.9	2000	0.900	41	1890.31	**40**	7.938
C2000.5	2000	0.500	182	1794.738	**178**	3413.501
C4000.5	4000	0.500	329	3198.524	**326**	37.46
MANN_a27	378	0.990	**4**	0.031	**4**	0.001
MANN_a45	1035	0.996	**4**	0.512	**4**	0.001
MANN_a81	3321	0.999	**4**	21.331	**4**	0.001
brock200_2	200	0.496	27	5.537	**26**	2.11
brock200_4	200	0.658	19	5.133	**18**	0.89
brock400_2	400	0.749	25	7.453	**24**	1.063
brock400_4	400	0.749	25	10.314	**24**	1.156
brock800_2	800	0.651	58	1621.504	**57**	1.843
brock800_4	800	0.650	59	118.132	**57**	13.365
gen200_p0.9_44	200	0.900	8	0.14	**7**	0.339
gen200_p0.9_55	200	0.900	**7**	1.048	**7**	0.016
gen400_p0.9_55	400	0.900	13	5.743	**12**	1.507
gen400_p0.9_65	400	0.900	12	2.405	**11**	0.6
gen400_p0.9_75	400	0.900	12	41.068	**11**	0.266
hamming8–4	256	0.639	**16**	12.969	**16**	0.141
hamming10–4	1024	0.829	35	83.906	**34**	8.912
keller4	171	0.649	20	10.041	**19**	27.201
keller5	776	0.752	44	1584.427	**42**	1296.893
keller6	3361	0.818	95	614.315	**92**	540.414
p_hat300–1	300	0.244	65	266.354	**65**	6.423
p_hat300–2	300	0.489	44	320.752	**42**	93.33
p_hat300–3	300	0.744	21	124.531	**19**	30.942
p_hat700–1	700	0.249	133	467.269	**132**	2454.861
p_hat700–2	700	0.498	90	965.563	**89**	115.964
p_hat700–3	700	0.748	41	654.411	**37**	30.106
p_hat1500–1	1500	0.253	257	1601.787	**254**	1629.413
p_hat1500–2	1500	0.506	169	1490.548	**168**	979.925
p_hat1500–3	1500	0.754	76	182.043	**67**	2332.43
DSJC500_5	500	0.502	57	18.346	**55**	59.062
DSJC1000_5	1000	0.500	100	3294.702	**99**	950.923

Table 10 Evolution of the best solution obtained by MSS (in the execution corresponding to Table 9) in the instances *keller4*, *keller5*, *p_hat300–2*, *p_hat300–3* and *p_hat700–1*

keller4		*keller5*		*p_hat300–2*		*p_hat300–3*		*p_hat700–1*			
Val.	*Time*	Val.	*Time*	Val.	*Time*	Val.	*Time*	Val.	*Time*	Val.	*Time*
22	*0.140*	51	*0.187*	48	*0.156*	23	*0.141*	172	*0.343*	141	*1.39*
21	*0.156*	48	*0.202*	47	*0.219*	22	*0.157*	155	*0.359*	140	*1.656*
20	*0.250*	47	*0.28*	46	*0.48*	21	*0.172*	154	*0.375*	139	*2.585*
19	*27.201*	46	*0.343*	45	*0.496*	20	*0.282*	153	*0.375*	138	*2.881*
		45	*0.781*	44	*0.824*	19	*30.942*	150	*0.5*	137	*4.741*
		44	*1.945*	43	*79.390*			147	*0.625*	136	*5.991*
		43	*9.352*	42	*93.300*			146	*0.859*	135	*16.581*
		42	*1296.893*					145	*0.922*	134	*17.487*
								143	*1.328*	133	*331.035*
								142	*1.375*	132	*2454.861*

Tables 10 and 11 show the evolution of the value of the best solution found by MSS (column "V*al.*") and the computation time used to obtain it (column "*Time*") for each of these instances. These values were obtained in the execution corresponding to Table 9. Specifically, Table 10 shows the evolution corresponding to the instances *keller4*, *keller5*, *p_hat300–2*, *p_hat300–3* and *p_hat700–1*. Table 11 shows the evolution corresponding to the instances *p_hat700–2 p_hat700–3*, *p_hat1500–1*, *p_hat1500*–2 and *p_hat1500–3*.

As an example, Figs. 8 and 9 show the graphs corresponding to the evolution of the *keller5* and *p_hat700–2* instances.

In Tables 10 and 11 and Figs. 8 and 9, the following can be observed:

– In instance *keller4*, MSS reaches a solution with 21 cliques in 0.156 seconds and with 20 cliques in 0.25 seconds. Meanwhile, GABC obtains the best solution with 21 cliques (over the 10 runs) in 0.82 seconds (the average

Table 11 Evolution of the best solution obtained by MSS (in the execution corresponding to Table 9) in the instances *p_hat700–2*, *p_hat700–3*, *p_hat1500–1*, *p_hat1500*–2 and *p_hat1500–3*

p_hat700–2		*p_hat700–3*		*p_hat1500–1*				*p_hat1500–2*		*p_hat1500–3*	
Val.	*Time*	Val.	*Time*	Val.	*Time*	Val.	*Time*	Val.	*Time*	Val.	*Time*
104	*0.313*	42	*0.906*	284	*1.015*	268	*15.238*	223	*0.781*	80	*0.637*
97	*0.328*	41	*1.062*	283	*1.015*	267	*22.072*	218	*0.797*	79	*0.701*
96	*0.672*	40	*2.237*	282	*1.031*	266	*22.087*	210	*0.812*	78	*0.737*
95	*0.735*	39	*4.299*	281	*1.187*	265	*28.808*	203	*0.828*	77	*1.003*
94	*1.844*	38	*7.503*	280	*1.187*	264	*31.69*	199	*0.843*	76	*2.378*
93	*9.902*	37	*30.106*	279	*2.437*	263	*34.66*	192	*0.859*	75	*4.206*
92	*13.095*			278	*2.922*	262	*38.165*	184	*0.875*	74	*8.847*
91	*46.167*			277	*3.234*	261	*39.399*	176	*0.89*	73	*10.769*
90	*115.293*			276	*3.25*	260	*77.743*	175	*1.062*	72	*24.396*
89	*115.964*			275	*4.815*	259	*121.491*	174	*1.062*	71	*35.917*
				274	*4.831*	258	*125.157*	173	*1.89*	70	*54.668*
				273	*5.159*	257	*128.244*	172	*13.976*	69	*106.732*
				272	*8.956*	256	*571.912*	171	*14.148*	68	*156.818*
				271	*10.019*	255	*575.079*	170	*175.211*	67	*2332.43*
				270	*12.894*	254	*1629.413*	169	*438.979*		
				269	*14.519*			168	*979.925*		

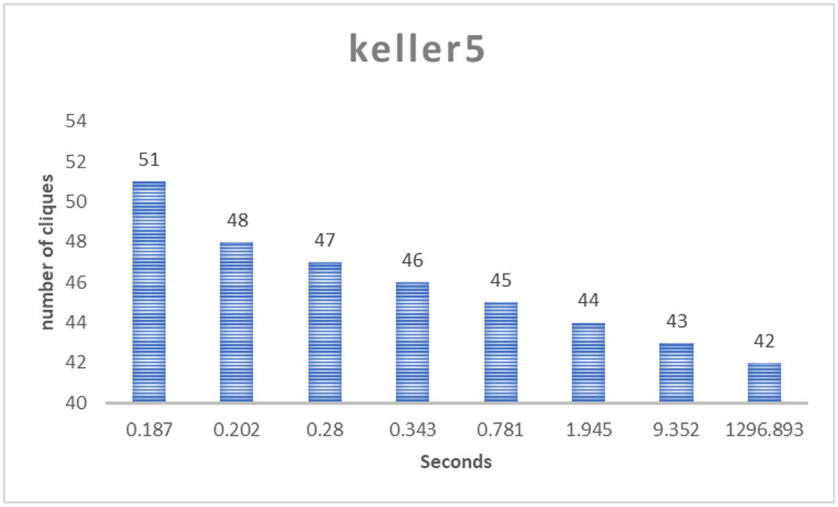


Fig. 8 Evolution of MSS in *keller5*

execution time), and SSGGA and MSTS reach solutions with 20 cliques in 2.14 and 10.041 seconds, respectively.

- In instance *keller5*, MSS reaches a solution with 47 cliques in 0.28 seconds and with 44 cliques in 1.945 seconds. Meanwhile, GABC reaches a solution with 47 cliques in 68.85 seconds, and SSGGA and MSTS reach solutions with 44 cliques in 39.98 and 1584.427 seconds, respectively.
- In instance *p_hat300–2*, MSS reaches a solution with 45 cliques in 0.496 seconds and with 44 cliques in 0.824 seconds. Meanwhile, SSGGA and GABC reach solutions with 45 cliques in 4.04 and 3.1 seconds, respectively, and MSTS reaches a solution with 44 cliques in 320.752 seconds.
- In instance *p_hat300–3*, MSS reaches a solution with 22 cliques in 0.157 seconds and with 21 cliques in 0.172 seconds. Meanwhile, SSGGA and GABC reach solutions with 22 cliques in 4.21 and 3.11 seconds, respectively, and MSTS reaches a solution with 21 cliques in 124.351 seconds.
- In instance *p_hat700–1*, MSS reaches a solution with 146 cliques in 0.859 seconds, with 135 cliques in 16.581 seconds and with 133 cliques in 331.035 seconds. Meanwhile, GABC reaches a solution with 146 cliques in 36.63 seconds, SSGGA reaches a solution with 135 cliques in 19.93 seconds, and MSTS reaches a solution with 133 cliques in 467.269 seconds.
- In instance *p_hat700–2*, MSS reaches a solution with 93 cliques in 9.902 seconds and with 90 cliques in 115.293 seconds. Meanwhile, SSGGA and GABC reach solutions with 93 cliques in 19.29 and 35.44 seconds, respectively, and MSTS reaches a solution with 90 cliques in 965.563 seconds.
- In instance *p_hat700–3*, MSS reaches a solution with 44 cliques in 0.437 seconds, with 43 cliques in 0.468 seconds and with 41 cliques in 1.062 seconds. Meanwhile,

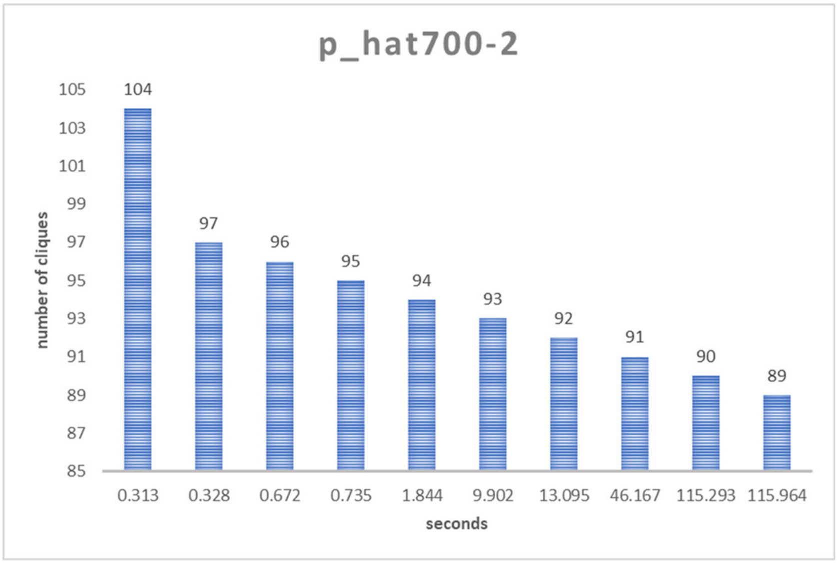


Fig. 9 Evolution of MSS in *p_hat700–2*

Table 12 CPUs used by the different heuristics

Method	CPU	Relative Speed (GHz)	Benchmarks			
			Gaming	*Desktop*	*Workstation*	Average
SSGGA and GABC [19]	Intel i5	3.5	*75*	*79*	*57*	75.2
MSTS [4]	Intel i7 (7700)	4.2	*79*	*83*	*66*	79.2
MSS	AMD 3990X	2.9	*81*	*81*	*146*	81.3

SSGGA reaches a solution with 44 cliques in 25.31 seconds, GABC reaches a solution with 43 cliques in 32.86 seconds, and MSTS reaches a solution with 41 cliques in 654.411 seconds.

– In instance *p_hat1500–1*, MSS reaches a solution with 284 cliques in 1.015 seconds, with 271 cliques in 10.019 seconds and with 257 cliques in 128.244 seconds. Meanwhile, GABC reaches a solution with 284 cliques in 263.04 seconds, SSGGA reaches a solution with 271 cliques in 120.12 seconds, and MSTS reaches a solution with 257 cliques in 1601.787 seconds.
– In instance *p_hat1500–2*, MSS reaches a solution with 176 cliques in 0.89 seconds, with 175 cliques in 1.062 seconds and with 169 cliques in 438.979 seconds. Meanwhile, GABC reaches a solution with 176 cliques in 254.52 seconds, SSGGA reaches a solution with 175 cliques in 103.6 seconds, and MSTS reaches a solution with 169 cliques in 1490.548 seconds.
– In instance *p_hat1500–3*, MSS reaches a solution with 80 cliques in 0.637 seconds, with 79 cliques in 0.701 seconds and with 76 cliques in 2.378 seconds. Meanwhile, SSGGA reaches a solution with 80 cliques in 160.47 seconds, GABC reaches a solution with 79 cliques in 290.49 seconds, and MSTS reaches a solution with 76 cliques in 182.043 seconds.

It should be noted, however, that the SSGGA and GABC times refer to the average times of the 10 complete runs corresponding to Table 7, and the solution value refers to the best solution obtained in these 10 runs. For MSTS, the solution value refers to the best solution obtained in the run corresponding to Table 9, and the time refers to the time needed to reach this solution.

In summary, from the above, it can be concluded that a) in most of the cases, our method obtains similar or strictly better solutions than the previous methods; b) in most of the cases, our method can reach solutions of the same quality as the final solutions obtained by the previous methods while using less total time than those methods (in some cases, two orders of magnitude less); and c) our method shows that it has the capacity to evolve, as it not only obtains good solutions quickly but is also able to improve them during their execution.

All the solutions obtained by MSS corresponding to Tables 7 and 9 can be found at the following link: www.ubu.es/metaheuristicos-grinubumet/ejemplos-y-datos-de-problemas. However, for a more rigorous comparison of the computation times, we provide Table 12 showing the CPUs used to run SSGGA, GABC, MSTS and our MSS, as well as their relative speeds. Also, we include some benchmarks that can be found at cpu.userbenchmark.com, (specifically those corresponding to the Gaming, Desktop and Workstation tests, as well as the average result).

5 Conclusions

Clique problems in graphs are interesting because they are attractive from a theoretical perspective and they have a wide range of applications. One of these problems is the so-called *Clique Partitioning Problem*, which has great applicability in different areas, such as timetabling, manufacturing, scheduling, and telecommunications. This is an NP-hard problem, and as described in this work, it can only be solved exactly in relatively small instances. Despite its practical importance, few resolution methods have been proposed for this problem in the recent literature. This work proposes a heuristic method that uses a tabu search procedure within a multistart strategy. An interesting characteristic of our method is that it allows unfeasible solutions to be visited. This gives flexibility to the exploration of the solution space, and in this way, the method achieves a dramatic improvement in both quality and computational time over the results of previous methods.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work was partially supported by FEDER funds and the Spanish State Research Agency (Projects PID2019-104263RB-C44 and PDC2021–121021-C22); the Regional Government of “Castilla y León”, Spain (Project BU071G19); the Regional Government of “Castilla y León”; and FEDER funds (Project BU056P20).

Declarations

Ethics approval This article does not contain any studies with human participants or animals performed by any of the authors.

Conflict of interest All authors declare that they have no conflicts of interest.

References

1. Allignol C, Barnier N, Gondran A (2012) Optimized flight level allocation at the continental scale. In: 5th international conference for research in air transportation, May 2012, Berkeley
2. Bhasker J, Samad T (1991) The clique-partitioning problem. Comput Math Appl 22(6):1–11
3. Blatt M, Wiseman S, Domany E (1996) Superparamagnetic clustering of data. Phys Rev Lett 76(18):3251–3254
4. Casado S, Laguna M, Pacheco J, Puche JC (2020) Grouping products for the optimization of production processes: a case in the steel manufacturing industry. Eur J Oper Res 286(1):190–202
5. Chen Z, Yuan L, Lin X, Qin L, Yang J (2020) Efficient maximal balanced clique enumeration in signed networks. In: Proceedings of The Web Conference 2020, pp 339–349
6. Dorndorf U, Jaehn F, Pesch E (2008) Modelling robust flight-gate scheduling as a clique partitioning problem. Transp Sci 42(3):292–301
7. Glover F, Laguna M (1997) Tabu Search. Kluwer Academic Press, London
8. Hu S, Wu X, Liu H, Li R, Yin M (2021) A novel two-model local search algorithm with a self-adaptive parameter for clique partitioning problem. Neural Comput & Applic 33(10):4929–4944
9. Katukuri M, Jagarapu M (2022) CIM: clique-based heuristic for finding influential nodes in multilayer networks. Appl Intell 52(5):5173–5184
10. Kim JT, Shin DR (2002) New efficient clique partitioning algorithms for register-transfer synthesis of data paths. J-Korean Phys Soc 40:754–758
11. Liang Z, Shan E, Kang L (2019) The clique-perfectness and clique-coloring of outer-planar graphs. J Comb Optim 38(3):794–807
12. Lü Z, Zhou Y, Hao JK (2022) A hybrid evolutionary algorithm for the clique partitioning problem. IEEE Trans Cybern 52(9):9391–9403
13. Ozcan E, Ersoy E (2005) Final exam scheduler-FES. In 2005 IEEE congress on. Evol Comput 2:1356–1363
14. Panda PR, Dutt ND, Nicolau A (2000) On-chip vs. off-chip memory: the data partitioning problem in embedded processor-based systems. ACM Trans Des Autom Electron Syst (TODAES) 5(3): 682–704
15. Pardalos PM, Xue J (1994) The maximum clique problem. J Glob Optim 4(3):301–328
16. San Segundo P, Artieda J (2015) A novel clique formulation for the visual feature matching problem. Appl Intell 43(2):325–342
17. San Segundo P, Lopez A, Batsyn M, Nikolaev A, Pardalos PM (2016) Improved initial vertex ordering for exact maximum clique search. Appl Intell 45(3):868–880
18. Schenker A, Last M, Bunke H, Kandel A (2003) Clustering of web documents using a graph model. In: Web Document Analysis: Challenges and Opportunities, pp 3–18
19. Sundar S, Singh A (2017) Two grouping-based metaheuristics for clique partitioning problem. Appl Intell 47(2):430–442
20. Terashima-Marin H, Ross P, Valenzuela-Rendon M (1999) Clique-based crossover for solving the timetabling problem with GAs. In: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). IEEE, vol 2, pp 1200–1206
21. Wu Q, Hao JK (2015) A review on algorithms for maximum clique problems. Eur J Oper Res 242(3):693–709
22. Xiao, S, Li, W, Yang, L, Wen, Z (2020) Graph-Coloring Based Spectrum Sharing for V2V communication. In 2020 International conference on UK-China emerging technologies (UCET), 1–4. IEEE

Joaquin Pacheco is currently Professor in Mathematics, Statistics and Operations Research at the University of Burgos, Spain. She received her Ph.D. in Mathematics (Operations Research) from the Complutense University of Madrid in 1994. His research interests include heuristic optimization and applications to real problems such as public transport, commercial logistic, sanitary logistics as well as deep learning models.

He has published more than 40 publications indexes by JCR. He has been the leader of different competitive National Projects supported by public entities. He has also carried out applied research work (research transfer) for several private entities.

Silvia Casado is currently Professor in Mathematics and Operations Research at the University of Burgos, Spain. She received her Ph.D. in Business (Operations Research) from the University of Burgos in 2003. The field of applications of her research includes social applications such as school transport, urban transport, location of medical facilities, classification methods applied to diagnosis of diseases, etc. She has published about 25 publications indexes by JCR and SJR. She has joined in different competitive National Projects supported by public entities such as Spanish Ministry of Education and Science, Ministry of Science and Innovation, Ministry of Economy and Competitiveness and Regional government of Castilla and León.