

A Penalty-Evaporation Heuristic in a Decomposition Method for the Maximum Clique Problem

by

Patrick St-Louis (stlouip@iro.umontreal.ca)
Jacques A. Ferland (ferland@iro.umontreal.ca)
Bernard Gendron (gendron@iro.umontreal.ca)

Département d'informatique et de recherche opérationnelle
Université de Montréal
P.O. box 6128, Succursale Centre-Ville
Montréal, Québec H3C 3J7

January, 2004

Abstract

In this paper, we present a heuristic method to solve the maximum clique problem, based on the concepts of penalty and evaporation. At each iteration, some vertex i is inserted into the current solution (always a clique) and the vertices that are not adjacent to vertex i are removed from the solution. The removed vertices are then penalized in order to reduce their potential of being selected to be inserted in the solution again during the next iterations. This penalty is gradually evaporating to allow vertices to become interesting subsequently. This penalty-evaporation heuristic method is embedded in a decomposition algorithm that restricts the search for a maximum clique to subgraphs, but performs an aggressive exploration of the feasible domain. Numerical results indicate that the penalty-evaporation heuristic method alone is effective and reliable, but the gain in quality obtained when embedding it in the decomposition algorithm is worthy of the additional computing time required.

1 Introduction

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E such that each edge in E connects a pair of vertices in V . When two vertices are connected by an edge, they are said to be adjacent. A clique in G is a subgraph where the vertices are all pairwise adjacent. In this paper we study the maximum clique problem which consists in finding a clique of maximum cardinality.

The maximum clique problem is underlying in several applications. Some interesting examples include coding theory, geometry, fault diagnosis, computer vision and pattern recognition (more details are available in [11]). It is well-known that the maximum clique problem is NP-hard. Hence there is no currently known polynomial algorithm to solve it exactly. There exist two main families of methods : exact methods that are often very slow but eventually find the optimal solution for small problems, and heuristic methods that can usually find very good solutions in reasonable time.

The exact methods are mostly branch-and-bound algorithms (see [2, 3, 4, 15, 20, 24, 26, 33, 34, 40, 42, 45, 46, 49] for some examples). Hence, they rely on lower and upper bounds which are usually computed using heuristic methods. It is possible to reduce the execution time of an exact algorithm by using parallel computing (see [41] for an example).

In contrast with exact methods that search extensively through all the feasible domain, a heuristic method can be seen as a strategy to search only in promising parts of it. In general, the more exhaustive is the search, the better the quality of the solution found is. On the other hand, a heuristic method cannot guarantee that the solution it finds is indeed the maximum clique, unless the size of the maximum clique is known in advance. But a heuristic method always provides a lower bound on the size of the maximum clique, since the solution found is feasible for the problem.

In this paper, we present a heuristic method to solve the maximum clique problem, based on the concepts of penalty and evaporation. At each

iteration, some vertex i is inserted into the current solution (always a clique) and the vertices that are not adjacent to vertex i are removed from the solution. The removed vertices are then penalized in order to reduce their potential of being selected to be inserted in the solution again during the next iterations. This penalty is gradually evaporating to allow vertices to become interesting subsequently. This penalty-evaporation heuristic method is embedded in a decomposition algorithm that restricts the search for a maximum clique to subgraphs, but performs an aggressive exploration of the feasible domain. Numerical results indicate that the penalty-evaporation heuristic method alone is effective and reliable, but the gain in quality obtained when embedding it in the decomposition algorithm is worthy of the additional computing time required.

We are using the DIMACS benchmark problems (available on the ftp site dimacs.rutgers.edu/pub/challenge) to analyse the efficiency of our algorithm. The numerical results indicate that our algorithm is competitive with the best heuristic methods. Indeed, the best known value is reached for 64 out of the 80 problems, and the average percentage difference between the best known value and the one generated with our method is 2.2% (note that the best known value is reached for 51 out of the 80 problems using one of the best known heuristic procedure [18] with an average percentage difference equal to 2.1%). These results are found using the same set of parameters for all instances. Better results can be found by adjusting parameters according to the density and shape of the graph. It is worthy to note that the penalty-evaporation approach has been also used to solve the graph coloring problem, and that the numerical results in [10] indicate that the approach is competitive with the best known heuristic methods for this problem.

The paper is organized as follows. In Section 2, we present a review of the relevant literature. In Section 3, we give a detailed description of the penalty-evaporation method. Then, we introduce the decomposition algorithm in Section 4. Experimental results for the DIMACS benchmark problems are presented and analyzed in Section 5. In particular, we indicate how we calibrate the penalty and evaporation parameters, and we compare the results generated with this calibrated version to some of the best known heuristics. Finally, Section 6 includes concluding remarks and future research avenues.

2 Literature Review

The maximum clique problem can be formulated as follows [11] :

(MCP)

$$\begin{aligned} \max \quad & \sum_{i=1}^{|V|} x_i, \\ \text{s.t.} \quad & x_i + x_j \leq 1 \quad , \forall (i, j) \notin E, & (2.1) \\ & x_i \in \{0, 1\} \quad , \forall i \in V. & (2.2) \end{aligned}$$

In this formulation, $x_i = 1$ if and only if vertex i belongs to the clique (solution). Hence, due to constraints (2.1), two vertices i and j cannot be in the same solution if they are not adjacent. Furthermore, since the objective function corresponds to the number of vertices in the solution, then the optimal solution is a clique of maximum cardinality. The formulation for the maximum independent set problem is similar to the MCP, but constraints (2.1) apply to all $(i, j) \in E$ instead. Hence, solving the maximum independent set problem is equivalent to solving the MCP in the complementary graph.

The continuous relaxation of this problem usually generates non-integer optimal solutions. For example, the solution $x_i = 0.5, \forall i \in V$ is a feasible solution for the relaxation, and the objective value is $\frac{|V|}{2}$. Therefore, for every graph having a maximum clique of size less than $\frac{|V|}{2}$, the relaxation of the maximum clique has a non-integer optimal solution.

The feasible domain of problem (MCP) is the set of all cliques in the graph G . Since the number of cliques in a graph can be exponentially large in $|V|$, an enumeration procedure is not practical (see Moon and Moser [37] for a study on the number of maximum cliques in a graph). Therefore, heuristic procedures are often used to provide near-optimal solutions without exploring all possible solutions of the feasible domain. Their main drawback is that they may be unable to move away from a local maximum to search for a global maximum.

Greedy sequential algorithms are popular heuristic methods. At each iteration, a greedy sequential algorithm chooses a vertex to insert in the so-

lution (clique), until no vertex left in the graph can improve the size of the current solution. MIN [30], one of the most straightforward greedy heuristic procedure, consists in selecting from the remaining subgraph the highest degree vertex, and deleting each vertex that is not adjacent to it. The algorithm stops when every vertex of the graph has been selected or deleted. When the algorithm stops, the set of selected vertices belongs to a large clique of the original graph.

These greedy algorithms are very fast, but fail to explore the entire solution space, thus they often miss the optimal solution. One way to avoid this drawback is to run the greedy algorithm on many different parts of the feasible domain (the term diversification is used to describe this strategy). Another drawback of greedy algorithms is their myopic behavior : they often fail to identify improving solutions that are very close (or “neighbors”) to the solution they obtained. A so-called intensification strategy is used to overcome this drawback, by allowing to search more extensively the neighboring solutions. Most modern heuristic methods include their own adapted diversification and intensification processes [29].

Aside from greedy algorithms, several heuristic methods (simulated annealing, tabu search, ascent) share the same basic process : at each iteration, they move from a current solution (feasible or not) to a nearby solution, trying to reach a better feasible solution. For instance, in the tabu search, we keep track of the most recently visited solutions in order to avoid revisiting them (i.e. to avoid cycling). But like in most heuristics, the tabu search may reach a local maximum from which it fails to escape. To avoid this behavior, Soriano and Gendreau [47] propose three diversification processes, one of which is probabilistic. Battiti and Protasi [7] propose a reactive tabu search where the search restarts periodically in different regions of the feasible domain. The selection of the new starting point depends on the previous searches (long term tabu list).

Population based methods rely on a population (or a set) of solutions evolving according to nature inspired processes in order to improve the quality of the solutions. Several of these are variants of the genetic algorithm (GA). Since it is well known that, in general, pure GAs can hardly compete with local search heuristics, several hybrid algorithms combining GA

and local search have been developed to take advantage of both approaches' strengths. The method proposed by Bui and Eppley [17] is in that spirit. Other methods take advantage of the problem structure to develop more efficient operators. This is illustrated in Marchiori [36], in Aggarwal et al. [1], in Balas and Niehaus [5], and in Bomze and Stix [14].

Neural network algorithms try to mimic the behavior of the human brain to solve difficult problems. Bertoni et al. [8] approximate the maximum clique problem using a sequence of Hopfield networks. Pelillo [43] relies on Motzkin and Straus [39] results to approximate the maximum clique problem on a relaxation labeling network. Bomze et al. [12] also rely on the results in [39] to develop a local search method based on the concept of replicator dynamics that has been successfully applied to the maximum weighted clique problem [13] and to the graph isomorphism problem [44].

The Motzkin-Straus formulation is also used to deal with the continuous relaxation of the integer programming formulation of the maximum clique problem (see [27, 28]). Busygin [19] also introduces a continuous-based heuristic generating very good results by restricting the feasible domain to a spherical form inside the standard unit cube. Finally, several heuristic methods take advantage of the specific structure of some classes of graphs (see for instance [6, 9, 16, 21, 23, 25, 31, 35, 38, 50]).

3 The Penalty-Evaporation Algorithm

3.1 Basic Strategy

The motivation for developing our penalty-evaporation (P-E) algorithm is to provide a method that could search a larger portion of the feasible domain than most standard greedy heuristic algorithms with the same computational efficiency. The numerical results indicate that the solutions generated with our calibrated version of the algorithm are very good (usually much better than MIN).

The P-E algorithm “moves” from a feasible solution to another by iteratively adding vertices to the solution and removing other vertices in such a way that the clique property is preserved at each iteration. The basic steps of the algorithm are summarized as follows :

- 1 : Find a feasible solution in the graph and use it as the initial solution.
- 2 : Iterate until a specified number of iterations without improvement is reached :
 - 2.1 : *Insertion*. Choose a vertex i to insert in the current solution.
 - 2.2 : *Removal*. Remove every vertex from the solution that is not adjacent to i (the current solution remains a clique).

Note that in Step 1, the initial clique can be identified by any fast clique-finding algorithm. However, to avoid being biased, a single vertex is randomly selected as the initial clique. To verify the robustness of the algorithm, two runs are performed for each instance in the testbed, as reported in Section 5.

To specify the selection mechanism in Step 2.1, we associate a value $V(i)$ with each vertex i and we choose the vertex i with the largest value $V(i)$. Intuitively, since the purpose is to find a large clique in the graph, it seems logical to insert the vertex having the largest number of adjacent vertices in the current solution. In the best case, the size of the current clique increases by one. Otherwise, since we remove the vertices that are not adjacent to it, this choice induces the smallest reduction of the size of the current clique. It follows that $V(i)$ should be proportional to $\bar{\delta}(i)$, the number of vertices adjacent to i in the current solution (the *current degree* of vertex i), i.e.,

$$V(i) \approx \bar{\delta}(i).$$

It is easy to see that if we select the vertex in Step 2.1 only according to the value $\bar{\delta}(i)$, then the procedure may cycle (i.e., inserting and removing sequentially a subset of vertices). Hence, a safeguard mechanism is required. Of course, such a mechanism could be to assign a tabu status to each removed vertex forbidding its selection for insertion during a fixed number of successive iterations. This alternative has the inconvenience of forbidding the selection of all removed vertices for some number of successive iterations.

But it is intuitively clear that we should allow earlier selection of removed vertices having larger current degree. Hence we use instead a *penalty factor* p to decrease the value $V(i)$ each time vertex i is removed in Step 2.2. Furthermore, at each subsequent iteration the *penalty* term $P(i)$ of vertex i evaporates according to an *evaporation factor* e . Thus the penalty $P(i)$ of vertex i is increased by the penalty factor p each time it is removed in Step 2.2 (i.e., $P(i) = P(i) + p$), and at each subsequent iteration, it is reduced by the evaporation factor e (i.e., $P(i) = \max\{0, P(i) - e\}$). Note that the penalty of a vertex is always non-negative. Thus, in our implementation, the values $V(i)$, updated at each iteration, are the following :

$$V(i) = \bar{\delta}(i) - P(i).$$

It should be obvious that the tabu status approach can be simulated using a very large penalty factor p and an evaporation factor $e = \frac{p}{nb}$ where nb corresponds to the fixed number of successive iterations where the selection of the removed vertex is forbidden. Note also the similarity with the approach used in ant colony search methods [22]. Indeed these constructive procedures rely on traces to select sequentially the variables and their values in order to generate better solutions. The values of the traces are adjusted according to the quality of the solutions generated, and they evaporate from iteration to iteration. The penalty term $P(i)$ can be seen as a (negative) trace in the selection process that slowly evaporates.

3.2 Tie-Breaking Criteria

The value of a vertex is the main criterion for choosing the next vertex in Step 2.1. But since the maximal value can be reached for several vertices, additional criteria are used to break ties. This process can be implemented in many ways; we chose to apply successively each criterion until the tie is broken. Hence, the order in which the criteria are applied has an impact on the resulting selection. Each ordering induces a different variant of our algorithm. The criteria are specified in terms of the following properties of each vertex i :

1. $\delta(i)$, the number of vertices adjacent to i in the graph G (i.e. the *degree* of the vertex);
2. $\bar{\delta}(i)$, the current degree of vertex i ;
3. $f(i)$, the number of times vertex i has been inserted in the solution since the beginning (i.e., the *frequency* of the vertex).

Recall that, in the context of our P-E algorithm, the probability of removing a vertex i from the current clique should, in general, decrease as $\bar{\delta}(i)$ increases. Now, in order to break ties in the selection process, we may have to choose between some vertex having a high current degree that has been selected often recently (i.e., having a high penalty term) and some other vertex having a low current degree that has not been selected recently (i.e., having a low penalty term). On the one hand, if the strategy is to explore more extensively the feasible domain, then the second vertex should be selected. Hence, the secondary criteria should lead to select the vertex having the smallest current degree and the smallest frequency in order to diversify the search. On the other hand, if the strategy is to intensify the search around some local minimum, then the first vertex should be selected. Hence, the secondary criteria should lead to select the vertex having the largest current degree and the largest frequency in order to insert the vertices having the highest potential of increasing the size of the current solution.

It follows from these comments that the criteria to break ties have to be selected according to the impact that we are seeking. Hence :

- 1) selecting the vertex with the largest degree leads the search to a part of the graph that is denser ;
- 2) diversification is induced by selecting the vertex with the smallest current degree or the smallest frequency ;
- 3) intensification is induced by selecting the vertex with the largest current degree or the largest frequency.

3.3 Outline of the Algorithm

Let G be the graph given as input to the P-E algorithm, along with the penalty and evaporation parameters, respectively p and e . Let CS and BS be the current and best solutions (respectively) found by the algorithm so far. The variable BS will contain the solution to output at the end of the execution. The P-E algorithm is summarized as follows :

- 1 : (Initialisation)
 - For each vertex j : $P(j) = 0$
 - Choose randomly a vertex i as the initial solution
 - $BS = CS = \{i\}$
 - For each vertex j :
 - If j is adjacent to i : $\bar{\delta}(j) = 1$
 - Else : $\bar{\delta}(j) = 0$
- 2 : While a specified number of iterations without improvement is not reached :
 - 2.1 : Evaporate : for each vertex j , $P(j) = \max\{P(j) - e, 0\}$
 - 2.2 : Construct the set of potential vertices
 - $S = \{j \notin CS \mid V(j) = \max_{l \notin CS} \{V(l)\}\}$
 - 2.3 : Select a vertex i in S using tie-breaking criteria
 - 2.4 : Insert i in CS . Remove from CS vertices that are not adjacent to i and add p to their penalty
 - 2.5 : Update the current degree of each vertex
 - 2.6 : If $|CS| > |BS|$ then $BS = CS$ (we improve the best solution)

4 The Decomposition Algorithm

4.1 Basic Strategy

The decomposition algorithm is an iterative procedure that makes use of an embedded maximum clique heuristic method (here, we use the P-E

algorithm). At each iteration, this heuristic method identifies a large clique, called *anchor*, in a residual graph obtained from the original graph by removing some of the vertices, along with their incident edges (initially, the residual graph is the original graph itself). We further apply the maximum clique heuristic method on the closed neighborhood of each vertex in the anchor (the closed neighborhood of vertex i consists of i itself along with its adjacent vertices and all their incident edges in the residual graph). If, for some vertex in the anchor, a larger clique is found, instead of looking at the closed neighborhoods of the other vertices in the anchor, we use the newly found clique as anchor and we start over this process, referred to as *aggressive search*. When no larger clique is found in any of the closed neighborhoods, we remove *all* vertices in the anchor to obtain a new residual graph and perform the next iteration. The algorithm stops when the residual graph is empty.

If the maximum clique heuristic method is effective at identifying large cliques, there is a high probability that no vertex in the anchor is part of a larger clique, but there is still a chance that some of these vertices have been mistakenly removed. To provide a *safeguard* against this type of error, we consider the set of all vertices (*including deleted ones*) that are adjacent to *every* vertex in the anchor and we find a large clique in that set by using the maximum clique heuristic method. This clique can be appended to the anchor to form a larger clique that becomes the new anchor, which is then deleted to generate the new residual graph.

We can view the decomposition approach as an auxiliary algorithm to the embedded maximum clique heuristic method, since its role is to help the heuristic method to produce better solutions than as a stand-alone algorithm. Indeed, the decomposition algorithm searches more extensively the entire feasible domain since every vertex of the graph is considered at least once. In particular, the aggressive search and safeguard steps implement mechanisms similar to the intensification features of modern heuristic techniques [29]. Indeed, intensification is partly achieved by looking for a better solution in the closed neighborhood of every vertex in the anchor. But, updating the anchor every time a better solution is found is an even more aggressive intensification mechanism. Similarly, by reconsidering previously deleted vertices, the safeguard step adds to the level of intensification. Note also that diversification is achieved by removing all vertices in the anchor and restarting the

search on the residual graph.

The decomposition algorithm should be contrasted with another variant that consists in removing one vertex at a time after searching its closed neighborhood (this kind of strategy is the basis of effective branching rules in exact branch-and-bound algorithms [2]). In addition to reducing the level of diversification, this other approach would hardly allow an efficient implementation of the safeguard step, since by iteratively removing vertices instead of cliques, the safeguard mechanism has to be applied either after searching the closed neighborhood of each vertex, thus introducing inefficiencies, or at regular intervals, but then reducing the effectiveness of the safeguard step. By contrast, the final anchor provided by the aggressive search is a natural target to efficiently apply the safeguard step.

4.2 Outline of the Algorithm

Let G be the original graph and let G' be the current residual graph after some vertices have been deleted. Let $G'(i)$ be the closed neighborhood of vertex i , i.e., the subgraph of G' induced by vertex i and every adjacent vertex of i . Finally, let BC denote the largest clique found by the algorithm so far, which will contain the solution to output at the end of the execution. The algorithm can be summarized as follows :

$G' = G, BC = \phi$

While G' is not empty, iterate :

- 1 : (Anchor selection) Find a large clique C in G' (using P-E)
- 2 : (Aggressive search) For each vertex i of C , iterate :
 - 2.1 : Find a large clique C' in $G'(i)$ (using P-E)
 - 2.2 : If $|C'| > |C|$ then $C = C'$ and restart at 2
- 3 (Safeguard) $S = \{i \in G \mid i \text{ is adjacent to every vertex in } C\}$
 - 3.1 : Find a large clique C_S in S (using P-E)
 - 3.2 : $C = C \cup C_S$
- 4 : (Update) If $|C| > |BC|$ then $BC = C$
- 5 : (Deletion) $G' = G' \setminus C$

Note that, in Step 2.1, the heuristic method could have been replaced by an exact algorithm to identify the maximum clique in $G'(i)$. It can be easily proven that in this case, BC contains an optimal solution at the end of the execution. However, the algorithm has been specifically designed to embed a heuristic method in Step 2.1. In particular, the safeguard step, 3, is only useful in that context.

5 Numerical Results

5.1 Parameters of the P-E Algorithm

In this section, we present the analysis to calibrate the parameters of our penalty-evaporation algorithm introduced in Section 3. The objective is to identify values for the parameters that seem appropriate for all graphs independently of their shape or their density. For every run of the algorithm, we set the number of iterations without improvement equal to the number of vertices in the graph.

5.1.1 The Penalty Factor

The penalty factor p is certainly the most important parameter of the procedure. Indeed, a proper value of p should assume that the algorithm spends an appropriate amount of time looking for a large clique in any part of the feasible domain before moving to another one.

On the one hand, if the value of p is too large, then the values of the vertices are reduced too fast and the diversification is completed prematurely before identifying an optimal solution in any part of the feasible domain. On the other hand, if the value of p is too small, the search is intensified in only a small portion of the feasible domain, thus making the search very dependent on the initial solution. Furthermore, since each graph has its own shape and

its own density, the value of the penalty factor should be fixed accordingly. If the graph is very dense, the algorithm does not need to move a lot throughout the graph; then the diversification is not an important issue, and a low value would be appropriate. But if the graph is sparse, a high value of p is required in order to search the feasible domain of the problem more extensively.

In order to identify an order of magnitude for the value of p , each of the 80 DIMACS problem is solved twice, using different starting solutions for the following values of $p : 0,1,2,5,10$. In these tests, the value of the evaporation factor e is fixed to 0, and no additional criterion is used to break ties (i.e. the vertex to be inserted is randomly selected among those having the largest value $V(i)$). Each entry in Table 1 is equal to the percentage number of times the algorithm generates the best known solution for some problem using the corresponding penalty factor p .

Penalty factor p	0	1	2	5	10
% number of times generating the best known solution	21%	38%	29%	23%	23%

TAB. 1 – Order of magnitude of the penalty factor p

The results in Table 1 indicate that the order of magnitude for p should be 1. This value can be seen as the starting point to identify the proper value of p for any specific graph.

5.1.2 The Penalty and Evaporation Factors

Similar numerical tests are completed to identify good values for the penalty and evaporation Factors, i.e., each of the 80 DIMACS instances is solved twice using different starting solutions for the pairs of values exhibited in Table 2. Note that no additional criterion is used to break ties, and that each entry in Table 2 is equal to the percentage number of times the algorithm generates the best known solution for some problem using the corresponding pair of values.

The results in Table 2 indicate that the quality of the results decreases as we move away from the order of magnitude of 1 for the penalty factor even if the value of the evaporation factor is adjusted. The results shown in bold characters in Table 2 allow to identify the best value for the evaporation factor associated with the corresponding values of the penalty factor. Since the role of the evaporation phase is to diminish a bit of the penalty's impact, it is only logical that the optimal penalty value increases as we set the evaporation factor to a higher value.

e	p	0.5	0.7	1.1	1.5	1.9	2.3	2.7	3.1	3.5
0.001		40.0	43.1	35.0	33.1	32.5				
0.005		45.6	40.0	38.1	35.6	33.8				
0.01		47.5	45.6	43.1	36.3	33.8				
0.02		39.4	45.6	45.0	41.3	34.4	30.6			
0.05		33.8	40.6	41.3	50.0	46.9	40.0	36.3		
0.075			35.0	43.1	46.9	50.0	43.1	42.5	41.3	
0.1			31.3	37.5	45.6	47.5	49.4	46.3	46.9	39.4
0.125					39.4	46.9	45.6	46.9	47.5	42.5
0.15					38.1	44.4	46.9	43.8	42.5	44.4
0.2							44.4	41.9	46.3	46.3
0.25							41.9	42.5	44.4	44.4
0.3									41.9	41.9

TAB. 2 – Pairs of penalty and evaporation factors (% number of times generating the best solution)

5.1.3 Tie-breaking Criteria

In this section we refer to the following tie-breaking criteria :

- Max δ among the set of vertices considered for insertion, select the one(s) having the largest degree δ .
- Max $\bar{\delta}$ among the set of vertices considered for insertion, select the one(s) having the largest current degree $\bar{\delta}$.
- Min $\bar{\delta}$ among the set of vertices considered for insertion, select the one(s) having the smallest current degree $\bar{\delta}$.
- Max f among the set of vertices considered for insertion, select the one(s) having the largest frequency f .
- Min f among the set of vertices considered for insertion, select the one(s) having the smallest frequency f .

The same testing approach is used to analyze the effectiveness of using a different tie-breaking criterion or pairs of criteria used successively. Table 3 summarizes the results for several combination strategies and for different pairs of values for the penalty and evaporation factors that seem better according to the results in Table 2. The entries in bold characters in each row of Table 3 correspond to the penalty-evaporation pair inducing the best performance for the corresponding strategy combination.

Referring to Table 3, it is interesting to note that there always exists a combination of strategies improving the performance of the algorithm for any pair of values for the penalty and evaporation factors. Furthermore, for the penalty-evaporation pair (1.1-0.02), all the strategies improve the performance of the algorithm. Also, for any combination of strategies, the performance of the algorithm is generally better using the penalty-evaporation pair (1.1-0.02).

It follows from this analysis that one of the best penalty-evaporation pair to use is (1.1-0.02) together with the tie-breaking strategy Max $\bar{\delta}$ and then Max δ . In the sequel, we are using these parameters and added Min f as a third tie-breaking criterion (whenever needed).

1 st crit.	2 nd crit.	$p - e$				
		0.7-0.01	1.1-0.02	1.5-0.05	1.9-0.075	2.3-0.1
Max δ		46.9	51.3	50.6	48.8	48.8
Max $\bar{\delta}$		45.0	48.1	51.3	49.4	48.1
Min $\bar{\delta}$		46.9	51.9	48.1	48.8	48.8
Min f		44.4	51.3	47.5	48.1	48.8
Max f		46.3	49.4	45.6	45.0	46.9
Max δ	Max $\bar{\delta}$	45.6	51.9	50.6	48.8	48.8
Max δ	Min $\bar{\delta}$	47.5	50.6	50.6	48.1	48.1
Max δ	Min f	47.5	50.6	48.8	46.9	48.1
Max δ	Max f	46.3	49.4	52.5	46.9	46.9
Max $\bar{\delta}$	Max δ	46.3	52.5	51.3	48.8	47.5
Max $\bar{\delta}$	Min f	44.4	50.6	48.1	47.5	49.4
Max $\bar{\delta}$	Max f	48.1	50.6	46.3	45.6	47.5
Min $\bar{\delta}$	Max δ	48.1	52.5	51.3	46.3	48.8
Min $\bar{\delta}$	Min f	47.5	49.4	47.5	46.3	46.3
Min $\bar{\delta}$	Max f	46.3	50.0	45.6	46.3	45.6
Min f	Max δ	50.0	50.0	49.4	48.8	48.8
Min f	Max $\bar{\delta}$	44.4	50.6	48.1	48.1	49.4
Min f	Min $\bar{\delta}$	44.4	50.6	48.1	48.8	48.8
Max f	Max δ	47.5	50.6	48.1	48.1	46.3
Max f	Max $\bar{\delta}$	46.3	50.0	45.6	45.6	47.5
Max f	Min $\bar{\delta}$	46.3	48.8	46.3	44.4	45.0

TAB. 3 – Tie-breaking criteria combinations (% number of times generating the best solution)

5.2 Numerical Comparisons

In this section, the 80 DIMACS problems are used to compare the performance of our methods with that of the following three methods :

- The MIN algorithm, as mentioned in Section 2, is one of the simplest and fastest heuristic procedure to find a large clique in a graph. Indeed, this algorithm finds a large clique in $O(|E|)$, where $|E|$ is the number of edges in the graph. But it is not very effective since it can determine the best known solution of only 18 out of the 80 DIMACS instances. Hence we can conjecture that the problems solved to their best known value by MIN should be the

easiest to solve using more sophisticated algorithms.

- The Qualex algorithm [18] is a continuous-based heuristic method to solve the maximum weighted independent set problem which is equivalent to the maximum weighted clique problem applied on the complementary graph. The problem is formulated as a quadratic programming problem where the feasible domain reduces to a unit hypercube. The edge constraints are embedded in the objective function. The approach proposed in Qualex is to restrict the search over a sequence of local spherical domains generated by moving the center of the sphere inside the unit hypercube. Furthermore, the eigenproperties of the matrix are used to set the parameters of the spheres. Qualex is a reliable algorithm, both because of its speed (in $O(a|V|^3)$, where a is the number of tried sphere centers) and because of the quality of the solutions generated (51 out of the 80 DIMACS instances are solved to their best known value, and the average proportion between the solution found by Qualex versus the best known value on the 80 DIMACS instances is equal to 97.9%).

- The Tabu search approach of Soriano and Gendreau [48] is very efficient to solve many difficult problems. The authors found three sets of parameters for which their tabu search algorithm was particularly successful, and they presented them at the DIMACS challenge [32]. We selected, for each graph, the best value obtained by those three versions of the tabu search algorithm. It is worthy to note that this combination generates the best known solution for 57 out of the 80 DIMACS problems.

The results given by these three procedures (MIN, Qualex and Tabu) are compared with those obtained with the penalty-evaporation procedure (P-E), with the penalty-evaporation procedure embedded in the decomposition algorithm (Decomp. + P-E), and with the MIN procedure embedded in the decomposition algorithm (Decomp. + MIN), respectively. The results are summarized in Tables 4 - 5, where each entry is equal to the size of the largest clique generated for the corresponding problem using the corresponding method. The second column (Opt) includes the optimal (or best known) values for the DIMACS problems. Furthermore, the values of the parameters in the P-E procedure are fixed to the values identified in Section 5.1. Again, two different starting solutions are used for the P-E algorithm ; whenever the size

of the largest clique found differs between the two runs, the corresponding values are separated by a comma. Also, for each problem, values identified in bold characters are equal to the optimal (or best known) value. It is interesting to note that for problems c1000_9 and c2000_9, better solutions than the previously best known solutions were generated during the calibration of the parameters for the P-E procedure (this is marked using * in Tables 4 - 5).

Problem	Opt	MIN	Decomp.	Qualex	Tabu	P-E	Decomp.
			+ MIN				+ P-E
brock200_1	21	19	19	21	21	20	20, 21
brock200_2	12	9	10	12	11	10	11
brock200_3	15	13	14	15	14	14,13	14, 15
brock200_4	17	15	15	17	16	16	17 ,16
brock400_1	27	22	23	27	25	23,24	25
brock400_2	29	21	23	29	25	23,24	29 ,25
brock400_3	31	21	23	31	25	24	31
brock400_4	33	23	23	33	25	23,24	25
brock800_1	23	17	19	23	21	19	21
brock800_2	24	18	19	24	21	19,20	21
brock800_3	25	18	19	25	21	19	22,21
brock800_4	26	18	20	26	21	20	21
c1000_9	68*	62	62	63	65	64,65	67
c125_9	34	31	33	33	34	34	34
c2000_5	16	13	15	16	16	15	16
c2000_9	77*	67	70	72	74	76,74	76
c250_9	44	42	42	43	44	44 ,43	44
c4000_5	18	14	16	17	17	16	18
c500_9	57	52	52	53	56	56,54	57
c-fat200-1	12	12	12	12	12	12	12
c-fat200-2	24	24	24	24	24	24	24
c-fat200-5	58	58	58	58	58	58	58
c-fat500-1	14	14	14	14	14	14	14
c-fat500-10	126	126	126	126	126	126	126
c-fat500-2	26	26	26	26	26	26	26
c-fat500-5	64	64	64	64	64	64	64
dsjc1000_5	15	13	14	14	15	14	15
dsjc500_5	13	11	12	13	13	12, 13	13
gen200_p0_9_44	44	37	39	39	44	44	44
gen200_p0_9_55	55	38	50	55	55	55	55
gen400_p0_9_55	55	48	49	50	54	51	53
gen400_p0_9_65	65	45	48	65	65	65	65
gen400_p0_9_75	75	44	52	75	75	75	75
hamming10-2	512	512	512	512	512	512	512
hamming10-4	40	36	36	36	40	40	40
hamming6-2	32	32	32	32	32	32	32
hamming6-4	4	4	4	4	4	4	4
hamming8-2	128	128	128	128	128	128	128
hamming8-4	16	16	16	16	16	16	16
johnson16-2-4	8	8	8	8	8	8	8
johnson32-2-4	16	16	16	16	16	16	16
johnson8-2-4	4	4	4	4	4	4	4
johnson8-4-4	14	14	14	14	14	14	14

TAB. 4 – Results for the 43 first DIMACS problems (size of largest clique)

Problem	Opt	Decomp.				Decomp.	
		MIN	+ MIN	Qualex	Tabu	P-E	+ P-E
keller4	11	11	11	11	11	11	11
keller5	27	23	25	26	27	26, 27	27
keller6	59	48	53	51	59	39,49	59
MANN_a27	126	125	125	126	125	125	125
MANN_a45	345	342	343	342	342	342	342
MANN_a81	1100	1096	1096	1096	1096	1096	1096
MANN_a9	16	16	16	16	16	16	16
p_hat1000-1	10	9	10	10	10	10	10
p_hat1000-2	46	43	45	45	46	46	46
p_hat1000-3	68	62	64	65	66	67, 68	68
p_hat1500-1	12	10	11	12	11	12,11	12
p_hat1500-2	65	62	63	64	65	65	65
p_hat1500-3	94	85	92	91	94	94	94
p_hat300-1	8	7	7	8	8	8	8
p_hat300-2	25	24	25	24	25	25	25
p_hat300-3	36	33	34	35	36	36	36
p_hat500-1	9	8	9	9	9	9	9
p_hat500-2	36	33	35	36	36	36	36
p_hat500-3	50	46	48	48	50	49, 50	50
p_hat700-1	11	8	9	11	11	11	11
p_hat700-2	44	42	44	43	44	44	44
p_hat700-3	62	59	62	61	62	62	62
san1000	15	10	10	15	10	8,9	15,10
san200_0_7_1	30	16	30	30	30	17	30
san200_0_7_2	18	15	15	18	18	13,15	18
san200_0_9_1	70	47	70	70	70	45,46	70
san200_0_9_2	60	38	60	60	60	60,43	60
san200_0_9_3	44	33	35	40	44	36,37	44
san400_0_5_1	13	8	13	13	13	8	13
san400_0_7_1	40	21	40	40	40	21,20	22,23
san400_0_7_2	30	17	22	30	30	18,19	30
san400_0_7_3	22	15	17	17	18	17	22
san400_0_9_1	100	92	100	100	100	54,55	100
sanr200_0_7	18	15	17	17	18	18	18
sanr200_0_9	42	37	41	41	42	42,41	42
sanr400_0_5	13	11	12	12	13	13,12	13
sanr400_0_7	21	18	20	20	21	21,20	21

TAB. 5 – Results for the 37 last DIMACS problems (size of largest clique)

	Opt	MIN	Decomp. + MIN	Qualex	Tabu	P-E	Decomp. + P-E
TOTAL	80	22.5	36.3	63.8	71.3	53.8,51.3	80.0,78.8
(w/o brocks)	68	26.5	42.6	57.4	82.4	63.2,60.3	89.7,88.2
Best of the 6 methods	80	23.8	38.8	65.0	73.8	56.3,52.5	83.8,82.5

TAB. 6 – % number of best solutions generated

The results in Table 6 show the percentage number of problems for which the corresponding methods either find the best known solutions for the problem (with and without the “brocks” family of graphs) or generate the best solution among the six methods.

	MIN	Decomp. + MIN	Qualex	Tabu	P-E	Decomp. + P-E
MIN	-	50	58	58	53	59
Decomp. + MIN	0	-	36	45	36	48
Qualex	0	6	-	22	22	27
Tabu	0	1	14	-	3	13
P-E	3	9	25	25	-	29
Decomp. + P-E	0	2	9	2	0	-

TAB. 7 – Number of times the top solver finds a larger clique than the left solver

In Tables 7 and 8, each pair of methods are compared. In Table 7, each entry is equal to the number of times the method of the corresponding column generates a better solution than the method in the corresponding row. For instance, the method Decomp. + MIN generates a better solution than MIN for 50 problems. For 27 problems, Decomp. + P-E generates better solutions than Qualex, but for nine others, Qualex generates better solutions than Decomp. + P-E.

In Table 8, each entry is equal to the difference of the corresponding entry and the symmetric one in Table 7. Hence entry (Decomp. + P-E, Qualex) = 18 and entry (Qualex, Decomp. + P-E) = -18. It follows that symmetric entries having small absolute values (close to 0) indicate a close performance of the corresponding methods, and the difference in performance grows when the absolute value of the symmetric entries increases. Thus the

		Decomp.			Decomp.	
		+			+	
	MIN	MIN	Qualex	Tabu	P-E	P-E
MIN	-	50	58	58	50	59
Decomp. + MIN	-50	-	30	44	27	46
Qualex	-58	-30	-	8	-3	18
Tabu	-58	-44	-8	-	-22	11
P-E	-50	-27	3	22	-	29
Decomp. + P-E	-59	-46	-18	-11	-29	-

TAB. 8 – Differences between each value in Table 7 and its symmetrical value

performance of P-E and Qualex methods is quite similar, but the Decomp. + P-E method seems to be much more effective than the MIN method.

The results in Table 8 indicate that the method MIN, even when embedded in the decomposition algorithm, is not competitive with the others as far as the quality of the solutions is concerned. Also, the other methods can be ordered in increasing order of their performance as follows : P-E, Qualex, Tabu, Decomp. + P-E. Note that this ordering is biased in favor of the Tabu search since the result for each problem in Tables 4- 5 is the best one among three runs of the method using different parameters. Furthermore, embedding a method in the decomposition approach increases greatly its performance (see the results of Decomp. + MIN versus MIN and of Decomp. + P-E versus P-E).

	r100.5	r200.5	r300.5	r400.5	r500.5
MIN	0.0	0.0	0.0	0.0	0.1
P-E	0.0	0.0	0.0	0.1	0.1
Decomp. + MIN	0.1	0.4	1.6	3.4	6.9
Qualex	0	1	3	8	20
Decomp. + P-E	0.2	1.6	4.1	8.6	18.9

TAB. 9 – Execution times, ordered from fastest to slowest (CPU secs.)

In Table 9, we order the methods in increasing order of their execution time when applied on the five benchmark instances used since the DIMACS challenge to compare the execution times of the algorithms. The values in

the table are the number of seconds each method used on an AMD Thunderbird 1200 MHz with 256M of memory. Each method has been compiled using Microsoft Visual C++ v6.0, except Qualex which has been compiled by Busygin and was graciously given for comparison purposes. Furthermore, we could not include the Tabu search in this table since the execution time (the total number of iterations allowed) is a parameter specified by the user. Finally, the results in Table 9 show that MIN and P-E have similar execution times, and the same is true for Qualex and Decomp. + P-E. We can also conclude that the P-E algorithm is at least as fast as any other method, and the quality of the results are at least as good as those obtained with MIN and Qualex.

Another issue is related to the degree of usefulness of the decomposition algorithm for graphs of various sizes and densities. In order to analyse this issue we apply the P-E method alone and the decomposition embedding the P-E method to randomly generated graphs of five different sizes (100 to 500) and having three different densities (0.5, 0.7, 0.9). For each pair of size and density, 20 graphs have been generated. Since each method is applied using two different seeds on every graph, we have the results of 40 runs of each method for any given pair of size and density. The values in Table 10 correspond to the percentage number of times embedding the P-E algorithm in the decomposition method leads to an improvement of the size of the largest clique found.

Density	Size	100	200	300	400	500
0.5		23%	35%	33%	63%	53%
0.7		33%	50%	63%	73%	78%
0.9		18%	65%	75%	83%	90%

TAB. 10 – % number of times embedding the P-E algorithm in the decomposition method improves the results

The results in Table 10 indicate that the number of times that the decomposition method is successful in improving the size of the clique generated seems to increase both with the size of the graph, and with its density. Since the clique generated using the P-E algorithm is strongly dependent of the initial solution, it should be obvious that the benefit of searching more ex-

tensively the feasible domain with the decomposition method increases with the size of the graph. Furthermore, since the size of the different subgraphs considered during the execution of the decomposition increases with the density of the graph, then, as the density of the graph gets closer to 1, applying this method becomes similar to a multistart of the P-E algorithm over the entire graph using different initial solutions. Thus the process increases the chances of finding a larger clique.

6 Conclusion

In this paper, we introduce a penalty-evaporation approach embedded into a decomposition algorithm to solve the maximum clique problem. The numerical results indicate that the effectiveness of the two methods MIN and P-E is greatly improved by embedding them in the decomposition algorithm. Furthermore the P-E method is competitive with Qualex, one of the most effective heuristic procedure to solve the maximum clique problem, while being also much faster.

The computing time of the decomposition algorithm could be improved by using parallelism to compute simultaneously Step 2.1 for each vertex i of the current largest clique. Additional analysis could also be done to identify stopping criteria to be used at Step 4 before the residual graph G' becomes empty. For instance, they could be based on the percentage of improvement or on the number of successive iterations where the best solution is not improved. The effectiveness of the penalty-evaporation method could be improved by adjusting the values of the parameters according to the shape and the density of the graph. Furthermore, it would be interesting to verify the effectiveness of variants of the penalty-evaporation method where a clique is inserted in Step 2.4 instead of a vertex or where the values $V(i)$ are integer (for instance, $V(i) = \bar{\delta}(i) - \lceil P(i) \rceil$) in order to use the tie-breaking criteria more extensively).

Acknowledgments

We are grateful to the anonymous referees whose comments have helped us write a better paper. We would like to thank S. Busygin for providing us his Qualex code and for allowing us to test it. Financial support for this project was provided by NSERC (Canada) and by FQRNT (Quebec). This support is gratefully acknowledged.

References

- [1] AGGARWAL, C., J.B. ORLIN and R. TAI, «Optimized crossover for the independent set problem», *INFORMS Operations Research*, vol. 45, 1997, pp. 226–234.
- [2] BABEL, L., «A fast algorithm for the maximum weight clique problem», *Computing*, vol. 52, 1994, pp. 31–38.
- [3] BABEL, L. and G. TINHOFFER, «A branch and bound algorithm for the maximum clique problem», *ZOR–Methods and Models of Operations Research*, vol. 34, 1990, pp. 207–217.
- [4] BALAS, E., S. CERIA, G. CORNUÉJOLS and G. PATAKI. «Polyhedral methods for the maximum clique problem». In *DIMACS series in discrete mathematics and theoretical computer science*, D. S. Johnson and M. Trick, Eds., vol. 26. American Mathematical Society, 11–13 Oct. 1996, pp. 11–28.
- [5] BALAS, E. and W. NIEHAUS, «Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems», *Journal of Heuristics*, vol. 4, no. 2, 1998, pp. 107–122.
- [6] BANG-JENSEN, J., J. HUANG and A. YEO, «Convex-round and concave-round graphs», *SIAM Journal on Discrete Mathematics*, vol. 13, no. 2, 2000, pp. 179–193.
- [7] BATTITI, R. and M. PROTASI, «Reactive local search for the maximum clique problem», *Springerlink Algorithmica*, vol. 29, no. 4, 2001, pp. 610–637.

- [8] BERTONI, A., P. CAMPADELLI and G. GROSSI. «A discrete neural algorithm for the maximum clique problem : Analysis and circuit implementation». In *Workshop on Algorithm Engineering (WAE97)* (Venice, Italy, 11–13 Sept. 1997), pp. 84–91.
- [9] BHATTACHARYA, B.K. and D. KALLER, «An $O(m + n \log n)$ algorithm for the maximum clique problem in circular-arc graphs», *Journal of Algorithms*, vol. 25, no. 2, 1997, pp. 336–358.
- [10] BLOEHLIGER, I. «A new heuristic for the graph coloring problem». Research report, École Polytechnique Fédérale de Lausanne, Département de Mathématiques Appliquées, 2001.
- [11] BOMZE, I.M., M. BUDINICH, P.M. PARDALOS and M. PELILLO. «The maximum clique problem». In *Handbook of Combinatorial Optimization*, D. Z. Du and P. M. Pardalos, Eds., vol. 4. Kluwer Academic Publishers, 1999, pp. 1–74.
- [12] BOMZE, I.M., M. BUDINICH, M. PELILLO and C. ROSSI, «Annealed replication : A new heuristic for the maximum clique problem», *Discrete Applied Mathematics*, vol. 121, 2002, pp. 27–49.
- [13] BOMZE, I.M., M. PELILLO and V. STIX, «Approximating the maximum weight clique using replicator dynamics», *IEEE Transactions on Neural Networks*, vol. 11, no. 6, 2000, pp. 1228–1241.
- [14] BOMZE, I.M. and V. STIX, «Genetic engineering via negative fitness : Evolutionary dynamics for global optimization», *Annals of Operations Research*, vol. 89, 1999, pp. 297–318.
- [15] BOURJOLLY, J.-M., G. LAPORTE and H. MERCURE, «A combinatorial column generation algorithm for the maximum stable set problem», *Operations Research Letters*, vol. 20, no. 1, 1997, pp. 21–29.
- [16] BROERSMA, H., T. KLOKS, D. KRATSCH and H. MULLER, «Independent sets in asteroidal triple-free graphs», *SIAM Journal on Discrete Mathematics*, vol. 12, no. 2, 1999, pp. 276–287.
- [17] BUI, T.N. and P.H. EPPLEY. «A hybrid genetic algorithm for the maximum clique problem». In *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA6)* (San Francisco, 15–19 July 1995), L. J. Eshelman, Ed., Morgan Kaufmann Publishers, pp. 478–484.
- [18] BUSYGIN, S. «Qualex 2.0 software release notes». Private communication, 2001.

- [19] BUSYGIN, S., S. BUTENKO and P.M. PARDALOS, «A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere», *Journal of Combinatorial Optimization*, vol. 6, no. 3, 2002, pp. 287–297.
- [20] CARRAGHAN, R. and P.M. PARDALOS, «An exact algorithm for the maximum clique problem», *Operations Research Letters*, vol. 9, 1990, pp. 375–382.
- [21] CICERONE, S. and G. Di STEFANO, «On the extension of bipartite to parity graphs», *Discrete Applied Mathematics*, vol. 95, no. 1-3, 1999, pp. 181–195.
- [22] DORIGO, M. and G.D. CARO. «The ant colony optimization meta-heuristic». In *New Ideas in Optimisation*, D. Corne, M. Dorigo, and F. Glover, Eds. McGraw-Hill, 1999, pp. 11–32.
- [23] DRAGAN, F.F., «Strongly orderable graphs - a common generalization of strongly chordal and chordal bipartite graphs», *Discrete Applied Mathematics*, vol. 99, no. 1-3, 2000, pp. 427–442.
- [24] FAHLE, T. «Simple and fast : Improving a branch-and-bound algorithm for maximum clique». In *Proceedings of the 10th Annual European Symposium (ESA2000)* (Rome, Italy, 2002), R. Möring and R. Raman, Eds., pp. 485–498.
- [25] GAVRIL, F., «Maximum weight independent sets and cliques in intersection graphs of filaments», *Information Processing Letters*, vol. 73, no. 5-6, 2000, pp. 181–188.
- [26] GENDREAU, M., J.C. PICARD and L. ZUBIETA, «An efficient implicit enumeration algorithm for the maximum clique problem», *Lecture Notes in Economics and Mathematical Systems*, vol. 304, 1988, pp. 70–91.
- [27] GIBBONS, L.E., D.W. HEARN and P.M. PARDALOS. «A continuous based heuristic for the maximum clique problem». In *DIMACS series in discrete mathematics and theoretical computer science*, D. S. Johnson and M. Trick, Eds., vol. 26. American Mathematical Society, 11–13 Oct. 1996, pp. 103–124.
- [28] GIBBONS, L.E., D.W. HEARN, P.M. PARDALOS and M.V. RAMANA, «Continuous characterizations of the maximum clique problem», *INFORMS Mathematics of Operations Research*, vol. 22, no. 3, 1997, pp. 754–768.

- [29] GLOVER, F. and M. LAGUNA, *Tabu Search*. Kluwer Academic, Boston, 1997.
- [30] HARANT, J., Z. RYJACEK and I. SCHIERMEYER, «Forbidden subgraphs and MIN-algorithm for independence number», *Discrete Mathematics*, vol. 256, no. 1-2, 2002, pp. 193–201.
- [31] HOCHBAUM, D., «Approximating clique and biclique problems», *Journal of Algorithms*, vol. 29, no. 1, 1998, pp. 174–200.
- [32] JOHNSON, D. S. AND M. TRICK, Eds., *Cliques, Coloring, and Satisfiability – Second DIMACS Implementation challenge*, vol. 26 of *DIMACS – Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 11–13 Oct. 1993.
- [33] MANNINO, C. and A. SASSANO, «An exact algorithm for the maximum stable set problem», *Computational Optimization and Application*, vol. 3, no. 4, 1994, pp. 243–258.
- [34] MANNINO, C. and A. SASSANO. «Edge projection and the maximum cardinality stable set problem». In *DIMACS series in discrete mathematics and theoretical computer science*, vol. 26. American Mathematical Society, 11–13 Oct. 1996, pp. 205–219.
- [35] MANNINO, C. and E. STEFANUTTI, «An augmentation algorithm for the maximum weighted stable set problem», *Computational Optimization and Application*, vol. 14, no. 3, 1999, pp. 367–381.
- [36] MARCHIORI, E. «A simple heuristic based genetic algorithm for the maximum clique problem». In *Proceedings of the 1998 ACM symposium on Applied Computing* (1998), ACM Press, pp. 366–373.
- [37] MOON, J.W. and L. MOSER, «On cliques in graphs», *Israel Journal of Mathematics*, vol. 3, 1965, pp. 23–28.
- [38] MOSCA, R., «Stable sets in certain P-6-free graphs», *Discrete Applied Mathematics*, vol. 92, no. 2-3, 1999, pp. 177–191.
- [39] MOTZKIN, T.S. and E.G. STRAUS, «Maxima for graphs and a new proof of a theorem of Turan», *Canadian Journal of Mathematics*, vol. 17, no. 4, 1965, pp. 533–540.
- [40] ÖSTERGÅRD, P.R.J., «A fast algorithm for the maximum clique problem», *Discrete Applied Mathematics*, vol. 120, 2002, pp. 195–205.
- [41] PARDALOS, P.M., J. RAPPE and M.G.C. RESENDE. «An exact parallel algorithm for the maximum clique problem». In *High Performance*

- Algorithms and Software in Nonlinear Optimization* (Ischia, Italy, 4–6 June 1997), R. D. Leone, A. Murli, P. M. Pardalos, and G. Toraldo, Eds., vol. 24 of *Applied Optimization*, Kluwer Academic Publishers, pp. 279–300.
- [42] PARDALOS, P.M. and G.P. RODGERS, «A branch and bound algorithm for the maximum clique problem», *Computers & Operations Research*, vol. 19, no. 5, 1992, pp. 363–375.
- [43] PELILLO, M., «Relaxation labeling networks for the maximum clique problem», *Journal of Artificial Neural Networks*, vol. 2, no. 4, 1995, pp. 313–328.
- [44] PELILLO, M., «Replicator equations, maximal cliques, and graph isomorphism», *Neural Computation*, vol. 11, no. 8, 1999, pp. 1933–1955.
- [45] RÉGIN, J.-C., «Using constraint programming to solve the maximum clique problem», *Lecture Notes in Computer Science*, vol. 2833, 2003, pp. 634–648.
- [46] ROSSI, C. «A replicator equation-based evolutionary algorithm for the maximum clique problem». In *Proceedings of the Congress on evolutionary computation (CEC2000)* (San Diego, CA, USA, 16–19 July 2000).
- [47] SORIANO, P. and M. GENDREAU, «Diversification strategies in tabu search algorithms for the maximum clique problem», *Annals of Operations Research*, vol. 63, 1996, pp. 189–207.
- [48] SORIANO, P. and M. GENDREAU. «Tabu search algorithms for the maximum clique problem». In *DIMACS series in discrete mathematics and theoretical computer science*, D. S. Johnson and M. Trick, Eds., vol. 26. American Mathematical Society, 11–13 Oct. 1996, pp. 221–242.
- [49] WOOD, D.R., «An algorithm for finding a maximum clique in a graph», *Operations Research Letters*, vol. 21, no. 5, 1997, pp. 211–217.
- [50] XUE, J., «Edge-maximal triangulated subgraphs and heuristics for the maximum clique problem», *Networks*, vol. 24, 1994, pp. 109–120.