A Fast Genetic Algorithm for the Max Cut-Clique Problem

Giovanna Fortez, Franco Robledo,
Pablo Romero, and Omar Viera

Instituto de Matemática y Estadística, IMERL
Facultad de Ingeniería - Universidad de la República
Montevideo Uruguay
(giovanna.fortez,frobledo,promero,viera)@fing.edu.uy,

Abstract. In Marketing, the goal is to understand the psychology of the customer in order to maximize sales. A common approach is to combine web semantic, sniffing, historical information of the customer, and machine learning techniques.
In this paper, we exploit the historical information of sales in order to assist product placement. The rationale is simple: if two items are sold jointly, they should be close. This concept is formalized in a combinatorial optimization problem, called *Max Cut-Clique* or *MCC* for short.
The hardness of the *MCC* promotes the development of heuristics. The literature offers a GRASP/VND methodology as well as an Iterated Local Search (ILS) implementation. In this work, a novel Genetic Algorithm is proposed to deal with the *MCC*. A comparison with respect to previous heuristics reveals that our proposal is competitive with state-of-the-art solutions.

Keywords: Marketing, Combinatorial Optimization Problem, Max Cut-Clique, Metaheuristics

1 Motivation

Large-scale corporations keep massive databases from their customers. Nevertheless, decision-makers and practitioners sometimes do not have the knowledge to combine Machine Learning techniques with Optimization, in order to exploit the benefits of Big Data.

An effective dialogue between academics and practitioners is crucial [7]. A bridge between the science-practice division can be found in Market Basket Analysis, or MBA [1]. In synthesis, MBA is a Data Mining technique originated in the field of Marketing. It has recent applications to other fields, such as bioinformatics [3, 5], World Wide Web [15], criminal networks [6] and financial networks [17]. The goal of MBA is to identify relationships between groups of products, items, or categories.

The information obtained from MBA is of paramount importance in the business strategy and operations. In Marketing, we can find valuable applications

such as product placement, optimal product-line offering, personalized marketing campaigns and product promotions. The analysis is commonly supported by Machine Learning, Optimization and Logical rules for association.

This work is focused on a specific combinatorial optimization methodology to assist product placement. Incidentally, it finds nice applications in biological systems as well. The problem under study is called Max Cut-Clique (MCC), and it was introduced by P. Martins [19].

Given a simple undirected graph $\mathcal{G} = (V, E)$ (where the nodes are items and links represent correlations), we want to find the clique $\mathcal{C} \subseteq V$ such that the number of links shared between $\mathcal{C}$ and $V - \mathcal{C}$ is maximized. Basically, the goal is to identify a set of *key-products* (clique $\mathcal{C}$), that are correlated with a massive number of products ($V - \mathcal{C}$). The MCC has an evident application to product-placement. For instance, the manager of a supermarket must decide how to locate the different items in different compartments. In a first stage, it is essential to determine the correlation between the different pairs of items, for psychological/attractive reasons. Then, the priceless/basic products (bread, rice, milk and others) could be hidden on the back, in order to give the opportunity for other products in a large corridor. Chocolates should be at hand by kids. Observe that the MCC appears in the first stage, while marketing/psychological aspects play a key role in a second stage for product-placement in a supermarket.

In [19], the author states that the MCC is presumably hard, since related problems such as $MAX - CUT$ and $MAX - CLIQUE$ are both $\mathcal{NP}$-Complete. A formal proof had to wait until 2018, where a reduction from $MAX - CLIQUE$ was provided [4]. Therefore, the MCC is systematically addressed by the scientific community with metaheuristics and exact solvers that run in exponential time. The first heuristic available in the literature on the MCC develops an Iterated Local Search [20]. Integer Linear Programming methods are also available [13]. The literature in the MCC is not abundant, since the problem has been posed recently.

A formal proof of complexity [4] is here included, since it is simple and it supports the applicability of heuristics. Furthermore, the main concepts of the previous GRASP/VND heuristic is given. A fair comparison between the novel proposal and this heuristic takes place.

This paper is organized in the following manner. Section 2 formally presents the hardness of the MCC. Section 3 briefly presents the previous GRASP/VND methodology. The novel Genetic Algorithm is introduced in Section 4. A fair comparison between both heuristics is presented using DIMACS benchmark in Section 5. Section 6 contains concluding remarks and trends for future work.

The reader is invited to consult the authoritative books on Graph Theory [14], Computational Complexity [10] and Metaheuristics [11] for the terminology used throughout this work.

2 Computational Complexity

The cornerstone in computational complexity is Cook's Theorem [8] and Karp reducibility among combinatorial problems [18].

Stephen Cook formally proved that the joint satisfiability of an input set of clauses in disjunctive form is the first $\mathcal{NP}$-Complete decision problem [8]. Furthermore, he provided a systematic procedure to prove that a certain problem is $\mathcal{NP}$-Complete. Specifically, it suffices to prove that the decision problem belongs to set $\mathcal{NP}$, and that it is at least as hard as an $\mathcal{NP}$-Complete problem. Richard Karp followed this hint, and presented the first 21 combinatorial problems that belong to this class [18]. In particular, $MAX - CLIQUE$ belongs to this list. The reader is invited to consult an authoritative book in Complexity Theory, which has a larger list of $\mathcal{NP}$-Complete problems and a rich number of bibliographic references [10].

Here, we formally prove that the MCC is at least as hard as $MAX - CLIQUE$. Let us denote $|\mathcal{C}|$ the cardinality of a clique $\mathcal{C}$, and $\delta(\mathcal{C})$ denotes the corresponding cutset induced by the clique (or the set) $\mathcal{C}$.

Definition 1 (MAX-CLIQUE).
GIVEN: a simple graph $G = (V, E)$ and a real number K.
QUESTION: is there a clique $\mathcal{C} \subseteq V$ such that $|\mathcal{C}| \geq K$?

For convenience, we describe MCC as a decision problem:

Definition 2 (MCC).
GIVEN: a simple graph $G = (V, E)$ and a real number K.
QUESTION: is there a clique $\mathcal{C} \subseteq G$ such that $|\delta(\mathcal{C})| \geq K$?

Theorem 1 was established for the first time in [4]. For a matter of completeness, the proof is here included.

Theorem 1. *The MCC belongs to the class of $\mathcal{NP}$-Complete problems.*

Proof. We prove that the MCC is at least as hard as $MAX - CLIQUE$. Consider a simple graph $G = (V, E)$ with order $n = |V|$ and size $m = |E|$. Let us connect a large number of M hanging nodes, to every single node $v \in V$. The resulting graph is called H (see Figure 1 for an example). If we find a polynomial-time algorithm for MCC, then we can produce the max cut-clique in H. But observe that the Max Cut-Clique $\mathcal{C}$ in H cannot include hanging nodes, thus it must belong entirely to G. If a clique $\mathcal{C}$ has cardinality c, then the clique-cut has precisely $c \times M$ hanging nodes. By construction, the cut-clique must maximize the number of hanging nodes, if we choose $M \geq m$. As a consequence, c must be the $MAX - CLIQUE$. We proved that the MCC is at least as hard as $MAX - CLIQUE$, as desired. Since MCC belongs to the set of $\mathcal{NP}$ Decision problems, it belongs to the $\mathcal{NP}$-Complete class. ■

Theorem 1 promotes the development of heuristics in order to address the MCC.

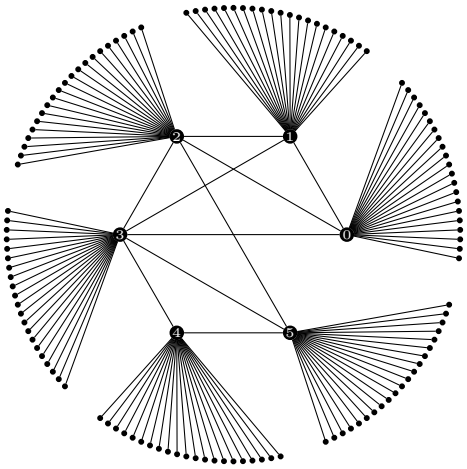

Fig. 1. Construction of H with $M = 21$ hanging nodes.

3 GRASP/VND Heuristic

GRASP and Tabu Search are well known metaheuristics that have been successfully used to solve many hard combinatorial optimization problems. GRASP is an iterative multi-start process which operates in two phases [22]. In the Construction Phase a feasible solution is built whose neighborhood is then explored in the Local Search Phase. Tabu Search [12, 2] is a strategy to prevent local search algorithms getting trapped in locally optimal solutions. A penalization mechanism called Tabu List is considered to avoid returning to previously visited solutions. For a complete description of these methods the reader is referred to the works of Glover and Laguna [12] and Resende and Ribeiro [22]. The reader is invited to consult the comprehensive Handbook of Metaheuristic for further information [11].

The full GRASP/VND implementation for the *MCC* is proposed in [4]. It strictly follows a traditional two-phase GRASP template with a Variable Neighborhood Descent (VND) as the local search phase, followed by an UPDATE of a Tabu list.

The goal in VND to combine a rich diversity of neighborhoods in order to obtain a solution that is both feasible and locally optimum for every feasible neighborhood. In [4], the authors consider five neighborhood structures considered to build the VND:

- **Remove**: a singleton $\{i\}$ is removed from a clique $\mathcal{C}$.
- **Add**: a singleton $\{i\}$ is added from a clique $\mathcal{C}$.
- **Swap**: if we find $j \notin \mathcal{C}$ such that $\mathcal{C} - \{i\} \subseteq N(j)$, we can include j in the clique and delete i (swap i and j).
- **Cone**: generalization of Swap for multiple nodes. The clique $\mathcal{C}$ is replaced by $\mathcal{C} \cup \{i\} - \mathcal{A}$, being $\mathcal{A}$ the nodes from $\mathcal{C}$ that are non-adjacent to i.

– **Aspiration**: this movement offers the opportunity of nodes belonging to the Tabu List to be added.

The previous neighborhoods take effect whenever the resulting cut-clique is increased. It is worth to remark that **Add**, **Swap**, and **Aspiration** were already considered in the previous ILS implementation [20]. The current VND is enriched with 2 additional neighborhood structures, named **Remove** and **Cone**. The Tabu list works during the potential additions during **Add**, **Swap** and **Cone**. On the other hand, **Aspiration** provides diversification with an *opportunistic unchoking* process: it picks nodes from the Tabu List instead. The reader is invited to find further details in [4].

4 Genetic Algorithm

Genetic Algorithms GA is a well-known family of metheuristics to solve hard combinatorial optimization problems. They belong to Evolutionary Computing, a wider family of metaheuristics. The goal is to emulate principles of the natural evolution of biological species, where the most adaptive individuals survive [16]. The reader can find a generous number of applications of GA to Engineering in [23].

Algorithm 1 GA: Basic Algorithm

Input: $\mathcal{G}$
Output: $\mathcal{C}$

1: $generation = 0$
2: $Pop(generation) \leftarrow Initialize(\mathcal{G})$
3: **while** ($generation <$ MAX_GENERATION) **do**
4: $\quad fitnessPop \leftarrow$ ***evaluate*** $(Pop(generation))$
5: $\quad Parents \leftarrow$ ***tournamentSelection*** $(Pop(generation))$
6: $\quad Offspring \leftarrow$ ***Crossover2x*** $(Parents)$
7: $\quad Offspring \leftarrow$ ***simpleMutation*** $(Offspring)$
8: $\quad newpop \leftarrow$ ***replacement*** $(Offspring, Parents)$
9: $\quad generation = generation + 1$
10: $\quad P(generation) = newpop$
11: **end while**
12: **return** $\mathcal{C}$ ▷ Best $\mathcal{C}$ Ever Found

In general, the algorithm randomly generates a set of feasible solutions, which are the *individuals* of the initial population. Then, combining crossover and mutation, this initial population evolves, and new generations are defined until a halting criterion is met. While the evolution is running, the exploration and exploitation of the solution space occurs, and an adequate operation selection must be applied [21]. Furthermore, the definition of the next generation trades between the selection of the best solutions and the preservation of the main characteristics of the population.

We strictly followed the traditional template of Genetic Algorithms, in Algorithm 1. In this case, the input is a simple undirected graph $\mathcal{G} = (V, E)$ and the result is a clique $\mathcal{C}$. The development is based on Malva Project, a collaborative and open source framework for computational intelligence in C++ [9]. The main reason of this choice is performance. The particular functions that are suitable for the *MCC* are detailed in the following subsections.

4.1 Fitness and Notation

Consider a simple graph $\mathcal{G} = (V, E)$. Let us sort the node-set $V = \{v_1, \ldots, v_n\}$. Each individual is represented by a bitmap, which is a binary word $\mathcal{X} = (x_1, \ldots, x_n)$ such that $x_i = 1$ means that node v_i belongs to the clique $\mathcal{C}$.

The *fitness* function is responsible for the survivability of the individuals in order to promote to the next generation. In this case, the fitness is precisely our objective function:

$$|\delta(\mathcal{C})| = \sum_{v \epsilon \mathcal{C}} deg(v) - |\mathcal{C}| \times |\mathcal{C} - 1| \tag{1}$$

The fitness evaluation is performed by Algorithm 2, where Expression (1) is considered.

Algorithm 2 EVALUATE

Input: $\mathcal{X}, \mathcal{G}$
Output: $fitness$

1: ***fitness*** $= 0$
2: **for** $i = 1$ **to** $|V|$ **do**
3: **if** $(\mathcal{X}[i]) = 1$ **then** $\forall i \in \mathcal{X}$
4: $fitness = fitness + deg(i)$
5: **end if**
6: **end for**
7: ***fitness*** $= fitness - |\mathcal{C}| * |\mathcal{C} - 1|$
8: **return *fitness***

Since we store the degree-sequence in a vector, the evaluation $deg(i)$ can be accessed fast, and using memory $O(1)$.

4.2 Selection: Tournament Selection

Selection determines which individuals will be parents of the next generation. In brief, the idea is to propagate good characteristics for the survival for the next generations. However, elitism selection must be avoided in order to prevent premature convergence of the method.

For that reason, the Goldberg Tournament Selection of size 2, plays a fundamental role in determining a trade-off between exploitation and exploration. Here, two individuals are randomly taken from the whole generation and their respective fitness compared, the one who maximizes this value is effectively selected as a parent of the new generation (Line 5, Algorithm 1).

4.3 Crossover

The evolutionary operators combine the genetic information of two or more individuals into new individuals with better fitness. Precisely, the recombination is executed by $Crossover2X$. Given two parents $\mathcal{X}$, $\mathcal{Y}$ and a crossover probability p_c, $Crossover2X$ returns two descendants $\mathcal{X}^*$ and $\mathcal{Y}^*$.

The operation takes place with probability p_c (Lines 1-2). In order to perform the crossover, 2 index-nodes $h < k$ are uniformly picked at random from the labels $\{1, \ldots, n\}$ (Line 3) and define two individuals, $\mathcal{X} = (x_1, \ldots, x_n)$ and $\mathcal{Y} = (y_1, \ldots, y_n)$. The children X^* and Y^* keep identical genetic information from their corresponding parents, for the elements $i < h$ (Lines 4-7) or $i \geq k$ (Lines 12-15), but a crossing occurs between the indices $i : h \leq i < k$ (Lines 8-11). Algorithm 3 returns the descendants $\mathcal{X}^*$ and $\mathcal{Y}^*$.

Algorithm 3 $Crossover2X$

Input: $\mathcal{X}$, $\mathcal{Y}$, $p_{crossover}$
Output: $\mathcal{X}^*$, $\mathcal{Y}^*$

1: $prob = random(0, 1)$
2: **if** $prob < p_c$ **then**
3: $(h, k) = random(0, |V| - 1)$
4: **for** $i = 1$ **to** $h - 1$ **do**
5: $\mathcal{X}^*[i] = \mathcal{X}[i]$
6: $\mathcal{Y}^*[i] = \mathcal{Y}[i]$
7: **end for**
8: **for** $i = h$ **to** $k - 1$ **do**
9: $\mathcal{X}^*[i] = \mathcal{Y}[i]$
10: $\mathcal{Y}^*[i] = \mathcal{X}[i]$
11: **end for**
12: **for** $i = k$ **to** $|V| - 1$ **do**
13: $\mathcal{X}^*[i] = \mathcal{X}[i]$
14: $\mathcal{Y}^*[i] = \mathcal{Y}[i]$
15: **end for**
16: **end if**

The traditional crossover operation considers a single index to cross the genetic information. Here, we used two indices for diversification. The parameter p_c is obtained using a statistical inference from a training set. Section 4.7 presents further details of the parametric selection.

4.4 Mutation

This technique applies an operation called mutation that works locally in a solution, changing one or more bits uniformly at random. In brief, consider a solution $\mathcal{X} = (x_1, \ldots, x_n)$. This operator performs a random walk in the neighborhood of $\mathcal{X}$. This is known as a *Simple Mutation*, which modifies a single bit from $\mathcal{X}$ at random, with probability p_m.

Algorithm 4 SIMPLE MUTATION

Input: $\mathcal{X}$, p_m
Output: $\mathcal{X}^*$

1: **for** $i = 1$ **to** $|V| - 1$ **do**
2: $\quad \mathcal{X}^*[i] = \mathcal{X}[i]$
3: **end for**
4: $prob = random(0, 1)$
5: **if** $prob < p_m$ **then**
6: $\quad k = random(0, |V| - 1)$
7: $\quad new\_k = random(0, 1)$
8: $\quad \mathcal{X}^*[k] = new\_k$
9: **end if**
10: **return** $\mathcal{X}^*$

Among the Evolutionary Computing models, GA applies crossover with a greater probability than mutation ($p_c > p_m$). The rationale behind this decision is that the exploration of the solution space is maximized. The adjustment of p_m is explained in Section 4.7.

4.5 Replacement: New generation

The replacement of the generation can be performed in various manners. Since our solution is diversity-driven, we selected the next generation using from the joint-set of descendants and parents (Line 8, Algorithm 1).

4.6 Feasible Solution

Observe that a bitmap can either represent a feasible or unfeasible solution. An unfeasible solution has different treatments according to the domain of the problem at hand. As the feasible solution must fulfill the adjacency-relation in the graph, every time its structure is manipulated by operations, a shaking algorithm is applied in order to preserve feasibility. Furthermore, the evaluation only can be practical over a feasible solution.

In this sense Algorithm 5 is a fundamental piece of code. It is inspired by the Construction Phase of the GRASP/VND introduced in [4].

Algorithm 5 FEASIBLE SOLUTION CONSTRUCTION

Input: $\mathcal{X}$, $\mathcal{G}$
Output: $\mathcal{X}$

$\mathcal{C} \leftarrow \emptyset, \quad \mathcal{C}' \leftarrow \emptyset$
$improving = \text{MAX\_ATTEMPTS}$
while $improving > 0$ **do**
  $i \leftarrow selectRandom(\mathcal{X})$
  $\mathcal{C}' \leftarrow [\mathcal{C} \cap N(i)] \cup \{i\}$
  if $|E'(\mathcal{C}')| > |E'(\mathcal{C})|$ **then**
    $\mathcal{C} \leftarrow \mathcal{C}'$
    $improving = \text{MAX\_ATTEMPTS}$
  else
    $improving = improving - 1$
  end if
end while
$\mathcal{X} \leftarrow \mathcal{C}$

4.7 Parameters Adjustment

A preliminary training set of instances were performed in order to tune the parameters using statistical analysis. Concretely, Rank Test was considered including a 30 independent runs set, finding optimal values in the cartesian product for p_m, p_c, *pop_size*.

The performance is measured in terms of the quality of the solution and computational efficiency. Each set of parameters was applied over DIMACS benchmark, for graphs with different link-densities. The selected instances for this preprocessing state were *p_hat300-1*, *MANN_a9* and *keller4*.

5 Computational Results

In order to test the performance of the algorithm, a fair comparison between both heuristics is carried out using DIMACS benchmark. The test was executed on an Intel Core i3, 2.2 GHz, 3GB RAM. Table 1 reports the performance of our GA for each instance [1].

All instances were tested using 60 runs and the stop criterion defined as $MAX\_GENERATION = 1000$, since the optimal value was reached from generation 250 to 850 in the numerous execution of the algorithm. Meanwhile calibration results for algorithm parameters was: $p\_mutex = 0.1$, $p\_crossover = 0.8$, $pop\_size = 200$. For the construction of a feasible solution in Algorithm 5, the value of $MAX\_ATTEMPTS = \lfloor \frac{|V|}{10} \rfloor * 2$.

Table 1 is divided into three main vertical areas for each instance. The leftmost one indicates the best solution known and reached according to [20]; then, the best solution reached and its computational time for [4] and the third

[1] All the scripts are available at the following URL: `https://drive.google.com/drive/folders/1mCTaJM4SA62rFhIutam1xDU-PZlXKtJF`

the results for GA. An additional column indicates the optimal gap between GRASP/VND and GA.

The reader can appreciate that for dense instances the GA maintains its computational times under 5 minutes in the worst case. In the cases where the best solution was reached at least one time over 60 runs, the optimal reported is close to the optimum value. Furthermore, a fair comparison between both approaches is conducted when averages values are reported.

Table 1. Comparative results.

ILS		**GRASP /VND**		**Genetic Algorithm**		**GAP**
Instances	$\|\delta(C)\|$	$\|\delta(C)\|$ avg	$T(s)$ avg	$\|\delta(C)\|$ avg	$T(s)$ avg	(%)
c-fat200-1	81	81	0.37	81	6.4	0.0
c-fat200-2	306	306	0.81	306	7.5	0.0
c-fat200-5	1892	1892	4.94	1892	12.5	0.0
c-fat500-1	110	110	2.46	110	16.15	0.0
c-fat500-2	380	380	5.83	380	14.3	0.0
c-fat500-5	2304	2304	10.85	2304	20.36	0.0
c-fat500-10	8930	8930	65.74	8930	32.59	0.0
p_hat300-2	4637	4636.2	3659.39	4633.40	171.9	0.0
p_hat300-3	7740	7726.8	3992.42	7387.27	279.8	0.04
keller5	15184	15183.24	1167.64	12382	50.57	0.18
c125_9	2766	2766	253.25	2737.2	5.0	0.01
MANN_a27	31284	31244.10	548.54	30405	46.49	0.03

The results described in this section reflect that our GA methodology is competitive with state-of-the-art solutions for the *MCC*, as well as having a quality solution and computational time efficiency. We underscore the accuracy reached and the considerable reduction in computational effort, finding optimal or near-optimal solutions in reasonable times.

6 Conclusions and Trends for Future Work

A deep understanding of item-correlation is of paramount importance for decision makers. Product-placement is a practical example in which the dialogue between operational researchers and decision makers is essential.

In this work we study a hard optimization problem, known as the Max Cut-Clique or *MCC* for short. Basically, the goal is to identify a set of *key-products*, that are correlated with a massive number of products. This idea has been successfully implemented in clothing stores. We believe that it is suitable for supermarkets and even for web sales.

Given the hardness of the *MCC*, a full Genetic Algorithm was introduced in order to solve the problem. A fair comparison with previous proposals reveals that our heuristic is both competitive and faster than a state-of-the-art solutions.

This fact provides a room for potential online applications, where the number of items and customers is dynamic.

As future work, we want to implement our solution into a real-life product-placement scenario. After the real implementation, the feedback of sales in a period is a valuable metric of success.

7 Acknowledgements

This work is partially supported by MATHAMSUD 19-MATH-03 Raredep, *Rare events analysis in multi-component systems with dependent components*, STICAMSUD 19-STIC-01 ACCON, *Algorithms for the capacity crunch problem in optical networks* and Fondo Clemente Estable *Teoría y Construcción de Redes de Máxima Confiabilidad*.

References

1. Herman Aguinis, Lura E. Forcum, and Harry Joo. Using market basket analysis in management research. *Journal of Management*, 39(7):1799–1824, 2013.
2. A. Amuthan and K. Deepa Thilak. Survey on tabu search meta-heuristic optimization. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 1539–1543, Oct 2016.
3. Gary D. Bader and Christopher W. V. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4:2, 2003.
4. Mathias Bourel, Eduardo A. Canale, Franco Robledo, Pablo Romero, and Luis Stábile. A GRASP/VND heuristic for the max cut-clique problem. In Giuseppe Nicosia, Panos M. Pardalos, Giovanni Giuffrida, Renato Umeton, and Vincenzo Sciacca, editors, *Machine Learning, Optimization, and Data Science - 4th International Conference, LOD 2018, Volterra, Italy, September 13-16, 2018, Revised Selected Papers*, volume 11331 of *Lecture Notes in Computer Science*, pages 357–367. Springer, 2018.
5. Sylvain Brohée and Jacques van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7(1):488, Nov 2006.
6. Gerben Bruinsma and Wim Bernasco. Criminal groups and transnational illegal markets. *Crime, Law and Social Change*, 41(1):79–94, Feb 2004.
7. Wayne F. Cascio and Herman Aguinis. Research in industrial and organizational psychology from 1963 to 2007: Changes, choices, and trends. *Journal of Applied Psychology*, 93(5):1062–1081, 2008.
8. Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.
9. Gabriel Fagundez. The malva project. a framework of artificial intelligence en c++. GitHub Inc. `https://themalvaproject.github.io`.
10. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, USA, 1979.

11. Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2nd edition, 2010.
12. Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.
13. Luis Gouveia and Pedro Martins. Solving the maximum edge-weight clique problem in sparse graphs with compact formulations. *EURO Journal on Computational Optimization*, 3(1):1–30, 2015.
14. F. Harary. *Graph Theory*. Addison Wesley series in mathematics. Addison-Wesley, 1971.
15. Monika Henzinger and Steve Lawrence. Extracting knowledge from the world wide web. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5186–5191, 2004.
16. John H. Holland. Adaption in natural and artificial systems. 1975.
17. Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Enumerating isolated cliques in synthetic and financial networks. In Boting Yang, Ding-Zhu Du, and Cao An Wang, editors, *Combinatorial Optimization and Applications*, pages 405–416, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
18. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
19. Pedro Martins. Cliques with maximum/minimum edge neighborhood and neighborhood density. 39:594–608, 03 2012.
20. Pedro Martins, Antonio Ladrón, and Helena Ramalhinho. Maximum cut-clique problem: ILS heuristics and a data analysis application. *International Transactions in Operational Research*, 22(5):775–809, 2014.
21. Colin R Reeves. Genetic algorithms. In *Handbook of metaheuristics*, pages 109–139. Springer, 2010.
22. Mauricio G.C. Resende and Celso C. Ribeiro. *Optimization by GRASP - Greedy Randomized Adaptive Search Procedures*. Computational Science and Engineering. Springer-Verlag New York, 2016.
23. Adam Slowik and Halina Kwasnicka. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, 2020.