Journal of Combinatorial Optimization manuscript No.
(will be inserted by the editor)

Efficient Algorithms for Cluster Editing

Lucas Bastos · Luiz Satoru Ochi · Fábio Protti · Anand Subramanian · Ivan César Martins · Rian Gabriel S. Pinheiro

Received: date / Accepted: date

Abstract The Cluster Editing Problem consists of transforming an input graph G into a *cluster graph* (a disjoint union of complete graphs) by performing a minimum number of *edge editing operations*. Each edge editing operation consists of either adding a new edge or removing an existing edge. In this paper we propose new theoretical results on data reduction and instance generation for the Cluster Editing Problem, as well as two algorithms based on coupling an exact method to, respectively, a GRASP or ILS heuristic. Experimental results show that the proposed algorithms are able to find high-quality solutions in practical runtime.

Keywords Combinatorial Optimization · Cluster Editing · Data Reduction · Metaheuristics · Exact Methods · Hybrid Algorithms

1 Introduction

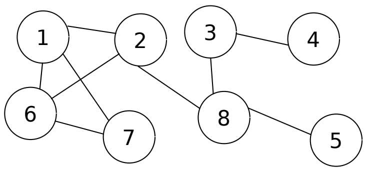
The Cluster Editing Problem (CEP) aims at finding the minimum number of *edge editing operations* that must be performed on a given input graph so that it becomes

L. Bastos
Financiadora de Estudos e Projetos (FINEP), Praia do Flamengo 200 - 1^{o}andar, Rio de Janeiro-RJ, 22210-065, Brazil

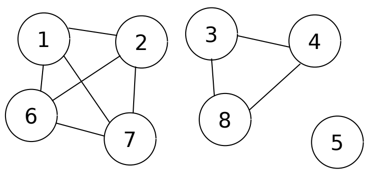
L. S. Ochi · F. Protti · I. C. Martins · R. G. S. Pinheiro
Universidade Federal Fluminense, Instituto de Computação, Rua Passo da Pátria 156, Bloco E - 3^{o}andar, São Domingos, Niterói-RJ, 24210-240, Brazil

A. Subramanian
Universidade Federal da Paraíba, Departamento de Engenharia de Produção, Centro de Tecnologia, Campus I - Bloco G, Cidade Universitária, João Pessoa-PB, 58051-970, Brazil
Tel.: +55 83 3216-7283
Fax: +55 83 3216-7549
E-mail: anand@ct.ufpb.br

a *cluster graph*, i.e., a disjoint union of complete graphs. Each edge editing operation consists of either adding a new edge or removing an existing edge. Figure 1 shows an example where two edge additions (edges (2,7) and (4,8)) and two edge deletions (edges (2,8) and (5,8)) transform an input instance (Fig. 1(a)) into a cluster graph (Fig. 1(b)). The CEP was firstly studied in Sen Gupta and Palit (1979), and proved to be NP-hard in Shamir et al (2004). Since then, several other works have studied the problem in many different contexts: exact methods (Böcker et al, 2009; Dehne et al, 2006; Rahmann et al, 2007), heuristics (Ben-Dor et al, 1999; Dehne et al, 2006; Rahmann et al, 2007; Sharan et al, 2003; Wittkop et al, 2007) and parameterized algorithms (Gramm et al, 2005; Protti et al, 2009). An Integer Linear Programming (ILP) formulation for the CEP was proposed in Grötschel and Wakabayashi (1989). Later, a 4-approximation algorithm was developed for a very simple and intuitive formulation called *MinDisAgree* (Charikar et al, 2005). This problem has drawn much attention due to applications in image processing and computational biology (Ben-Dor et al, 1999; Hartuv et al, 2000; Jain et al, 1999; Milosavljevic et al, 1995; Sharan et al, 2003; Tatusov et al, 2003; Wittkop et al, 2007), among other areas.



(a) Instance.



(b) Feasible Solution.

Fig. 1 Cluster Editing Problem: example.

Motivated by the large interest in the CEP, as well as its wide applicability, we propose in this work new theoretical and algorithmic developments for the unweighted variant of the problem, where there are no weights associated with edges. The main contributions of this work are: (i) theoretical results related to data reduction and instance generation; (ii) new GRASP- and ILS-based heuristics for the problem; (iii) two algorithms consisting of coupling an exact method based on Set Partitioning to the mentioned heuristics.

The remainder of this work is organized as follows. Section 2 presents the necessary notation, terminology and theoretical background. Section 3 states new theoretical results which are useful to understand instance behavior and improve algorithms. The ILP formulation used to exactly solve instances is detailed in Section 4. In Section 5, a reduction rule is proposed based on the previous theoretical results presented in Section 3; other reduction rules previously known in the literature and employed in this work are also described. Sections 6 and 7 describe the proposed heuristics based, respectively, on Greedy Randomized Adaptive Search Procedure (GRASP) and Iterated Local Search (ILS), while Section 8 explains the GRASP + Set Partitioning and ILS + Set Partitioning approaches. A more detailed description of these heuristics can be found in Bastos (2012). Section 9 discuss techniques for generating random instances for the CEP and estimating the difficulty of these instances. Finally, Section 10 shows the experimental results, where both reduction rules and algorithms are evaluated. Section 11 contains our concluding remarks.

2 Background

The graphs dealt with in this work are finite and simple. The vertex set and the edge set of a graph G are denoted by $V(G)$ and $E(G)$, respectively. We assume $V(G) = \{1, 2, \ldots, n\}$. The set of neighbors of a vertex i is denoted by $N(i)$, and the set of non-neighbors of i by $\overline{N}(i)$; in addition, define $N[i] = N(i) \cup \{i\}$. If $j \in \overline{N}(i)$ then we say that ij is a *non-edge* of G. The subset of neighbors (resp. non-neighbors) of i in a subset $S \subseteq V$ is denoted by $N(i, S)$ (resp. $\overline{N}(i, S)$). If G is a graph and $V' \subseteq V(G)$, we denote by $G - V'$ the graph derived from G by removing all vertices of V'. A graph H is an *induced subgraph* of a graph G if $V(H) \subseteq V(G)$ and the following property holds: if $ij \in E(G)$ and $i, j \in V(H)$ then $ij \in E(H)$. A *clique* is a subset of pairwise adjacent vertices. A *complete graph* is a graph whose vertex set is a clique. The *disjoint union* of graphs $G_1, \ldots, G_q$ is the graph with vertex set $\cup_{j=1}^{q} V(G_j)$ and edge set $\cup_{j=1}^{q} E(G_j)$. A *cluster graph* is a disjoint union of complete graphs; the cliques that define the vertex sets of such complete graphs are called *clusters*. An *isolated vertex* is a vertex with no neighbors. The *distance* between two vertices $i, j \in V(G)$, denoted by $d_G(i, j)$, is the minimum number of edges of a path linking i and j in G. If there is no path linking i and j in G then $d_G(i, j) = \infty$. The *diameter* of a graph G is defined as $diam(G) = \max\{d_G(i, j) \mid i, j \in V(G)\}$. An *edge editing operation* on a pair (i, j) of vertices of a graph G is an operation that consists of either *deleting* edge ij (if $ij \in E(G)$) or *adding* edge ij (if $ij \notin E(G)$). In the former case it is an *edge deletion*, and in the latter *edge addition*.

For a graph G, CEP(G) denotes the CEP with input G. We say that G' is a *feasible* or *candidate solution* of CEP(G), or simply a *solution*, if $V(G') = V(G)$ and G' is a cluster graph. For two sets S, S', denote $S \triangle S' = (S \setminus S') \cup (S' \setminus S)$. The *cost* of a solution G' is defined as $cost(G') = |E(G) \triangle E(G')|$. In other words, $cost(G')$ is the number of edges of G that are converted into non-edges of G', plus the number of non-edges of G that are converted into edges of G'. We define $opt(G) = \min\{cost(G') \mid G' \text{ is a solution of CEP}(G)\}$. A solution G' of CEP(G) is *optimal* if $cost(G') = opt(G)$. According to this terminology/notation, the problem dealt with in this work can be stated as follows: given a graph G, find an optimal solution G' of CEP(G).

The cost of a solution can be alternatively defined as follows. Given a graph G, the cost of a vertex $i \in V(G)$ relative to a subset $S \subseteq V(G)$ is defined by $cost^{+}(i, S) = |N(i, S)|$. Define also $cost^{-}(i, S) = |\overline{N}(i, S)|$. The cost of a subset $S \subseteq V(G)$ is given by $cost(S) = \sum_{i \in S} cost^{-}(i, S) + \sum_{i \in S} cost^{+}(i, V(G) \setminus S)$. The idea behind this notation is as follows: if a subset S induces a dense subgraph and defines a small edge cut $(S, V(G) \setminus S)$ then $cost(S)$ tends to be small. Note that the cost of a solution G' formed by clusters $C_1, \ldots, C_q$ is given by $cost(G') = \frac{1}{2} \sum_{j=1}^{q} cost(C_j)$. The *size* of a solution G' is denoted by $|G'|$ and is equal to its number of clusters.

Denote by P_n the graph consisting of a path on n vertices. It is a simple exercise to show that G is a cluster graph if and only if G does not contain P_3 as an induced subgraph. The graph P_3 is also called a *conflict*. Note that CEP(G) can also be stated as follows: find the minimum number of edge editing operations that "solve" all the conflicts. Let ijk be a conflict in G; it can be solved by either adding edge ik or deleting one of the edges ij, jk.

3 Theoretical Results

In this section we present some new theoretical results that will be useful for our discussions and algorithms. The first result (Proposition 1) says that vertices that are at distance three or more in a graph G cannot be put in a same cluster in any optimal solution of CEP(G). Algorithms may use this fact in order to limit the search space of cluster assignment possibilities for the vertices, especially for sparse instances, where considerable percentages of pairs (i, j) are expected to satisfy $d_G(i, j) \geq 3$. The second result (Theorem 1) says that CEP(G) is still hard even when G has diameter two, that is, when Proposition 1 cannot help to limit the search space of the problem.

Proposition 1 *Let G be a graph and let $i, j \in V(G)$ such that $d_G(i, j) \geq 3$. Then in any optimal solution of CEP(G) vertices i and i belong to distinct clusters.*

Proof. Let G' be an optimal solution of CEP(G), and suppose by contradiction that i, j belong to a same cluster $C \subseteq V(G') = V(G)$. Since i and i have no common neighbors in G, cluster C can be partitioned as follows: $C = C_0 \cup C_i \cup C_j \cup \{i, j\}$, where:

$$C_0 = \{k \in C\backslash\{i, j\} \mid ik, jk \notin E(G)\},$$
$$C_i = \{k \in C\backslash\{i, j\} \mid ik \in E(G), jk \notin E(G)\},$$
$$C_j = \{k \in C\backslash\{i, j\} \mid ik \notin E(G), jk \in E(G)\}.$$

Assume without loss of generality that $|C_i| \geq |C_j|$. Consider another solution G'' of CEP(G), obtained from G' by removing the subset of edges given by

$$E' = \{jk \in E(G') \mid k \in C\backslash\{j\}\}.$$

Note that j is an isolated vertex in G''. Now we compare $cost(G')$ with $cost(G'')$. We analyze only edge editing operations involving pairs of vertices (k, ℓ) such that $\{k, \ell\} \cap \{i, j\} \neq \emptyset$ (note that other edge editing operations contribute the same amount to $cost(G')$ and $cost(G'')$). Then it follows that

$$cost(G') - cost(G'') = 2|C_0| + |C_i| + |C_j| + 1 - (|C_0| + 2|C_j|) = |C_0| + |C_i| - |C_j| + 1.$$

This implies that $cost(G') - cost(G'') > 0$, a contradiction. ■

Theorem 1 *The Cluster Editing Problem is NP-hard even restricted to graphs of diameter two.*

Proof. The proof consists of a reduction from the CEP for general graphs, proved to be NP-hard in Shamir et al (2004). Let G be an instance with n vertices for the CEP. We construct a graph G' of diameter two such that $opt(G) \leq K$ if and only if $opt(G') \leq K + n$. Define $V(G') = V(G) \cup C$, where C is a subset of $n+1$ new vertices, i.e., $|C| = n+1$ and $C \cap V(G) = \emptyset$. Choose a vertex $c \in C$ and define

$$E(G') = E(G) \cup \{ic \mid i \in V(G)\} \cup \{c'c'' \mid c', c'' \in C, c' \neq c''\}.$$

In other words, graph G' is constructed by adding to G a clique C containing a special vertex c linked to all vertices in G. Note that $d_{G'}(i,j) \leq 2$ for all $i, j \in V(G')$.

Let H be a solution of CEP(G) satisfying $cost(H) \leq K$. A solution H' of CEP(G') satisfying $cost(H') \leq K + n$ can be easily obtained by performing the same edge editing operations as in H, plus n edge deletions involving pairs (i, c) for $i \in V(G)$.

Conversely, let H' be a solution of CEP(G') satisfying $cost(H') \leq K + n$. We first construct from H' another solution of CEP(G') with cost not greater than $cost(H')$, as follows:

(i) Let H_1 be the graph obtained from H' by repeatedly applying the following operation, while applicable: if there is a clique D in H' such that

$$c \notin D, D \cap V(G) \neq \emptyset, \text{ and } D \cap C \neq \emptyset$$

then remove the subset of edges $\{ij \in E(H') \mid i \in D \cap V(G)),\ j \in D \cap C\}$ (in other words, split D into cliques D_1 and D_2 such that $D_1 = D \cap V(G)$ and $D_2 = D \cap C$). Since all the edges removed in this step are not present in G', we have $cost(H_1) \leq cost(H')$.

(ii) Let H_2 be the graph obtained from H_1 by performing the following edge additions: if there are distinct cliques $D_1, D_2, \ldots, D_q$ in H_1 such that $D_i \subseteq C$ for $i = 1, \ldots, q$, add the subset of edges $\{c'c'' \mid c' \in D_i,\ c'' \in D_j, 1 \leq i < j \leq q\}$ (i.e., group all cliques D_i into a single one). Since all the edges added in this step are all in G', we have $cost(H_2) \leq cost(H_1)$.

(iii) Let H_3 be the graph obtained from H_2 by performing the following edge editing operations: if c is in a clique D such that $D \setminus \{c\} \subseteq V(G)$, remove all the edges in the subset $\{ic \mid i \in D \setminus \{c\}\}$, and add all the edges in the subset $\{cc' \mid c' \in C \setminus \{c\}\}$. Note that the edges added are all in G'. In addition, in any situation, the number of edges added in this step is greater than or equal to the number of edges removed (if $D \setminus \{c\} \neq \emptyset$ then at most n edges are removed and exactly n edges are added). Therefore, $cost(H_3) \leq cost(H_2)$.

After applying steps (i), (ii), and (iii) above, $cost(H_3) \leq cost(H')$. Also, C is one of the cliques in H_3, i.e., n edge deletions involving pairs (i, c) for $i \in V(G)$ are accounted for in $cost(H_3)$. Thus $H = H_3 - C$ is a solution of CEP(G) such that $cost(H) \leq K$. ■

4 Mathematical Formulation

We briefly describe below the ILP formulation used in this work to solve exactly the CEP, proposed in Charikar et al (2005). It relies on the simple fact that a graph G is a cluster graph if and only if G does not contain the graph P_3 (a path formed by three vertices) as an induced subgraph. Recall that $V(G) = \{1, \ldots, n\}$. For two vertices i, j with $i < j$, let x_{ij} be a binary variable such that $x_{ij} = 0$ if and only if vertices i and j belong to the same clique in a final solution.

$$\text{minimize} \quad \sum_{i<j,\ ij\ \in E(G)} x_{ij} + \sum_{i<j,\ ij\ \notin E(G)} (1 - x_{ij}) \tag{1}$$

$$\begin{aligned} &\text{subject to} \\ &x_{ik} \leq x_{ij} + x_{jk}\ , \\ &x_{ij} \leq x_{ik} + x_{jk}\ , \\ &x_{jk} \leq x_{ij} + x_{ik} \qquad \text{for all } i < j < k \end{aligned} \tag{2}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for all } i < j \tag{3}$$

Note that the objective function (1) minimizes the number of edges that are converted into non-edges plus the number of non-edges that are converted into edges. There are $O(n^3)$ triangle inequalities (2) that eliminate the induced subgraphs isomorphic to P_3.

5 Reduction Rules

In view of Proposition 1, a new reduction rule can be applied in order to reduce the search space of the CEP. In this work, we evaluate the impact of this rule for the exact algorithm in Section 10.2.

Rule 1. If $d_G(i, j) \geq 3$ then set ij as a *permanent non-edge*.

A *permanent non-edge* is a non-edge that cannot be edited, that is, cannot be converted into an edge of a solution. *Permanent edges* are defined similarly. Rule 1 is static, in the sense that it can be applied to graph G in a pre-processing step, independently of previous choices of edge editing operations. For the integer linear program (1)-(3), it implies to set $x_{ij} = 1$ permanently.

Rule 1 can be implemented to run in time complexity of matrix multiplication, since the ij-entry of matrix $(A_G + I)^2$ is zero if and only if vertices i and j are at distance three or more, where A_G is the adjacency matrix of G and I is the identity matrix.

Other well-known rules (Böcker et al, 2009; Gramm et al, 2005) have been employed in this work:

Rule 2. If i and j are true twins (i.e., $N[i] = N[j]$) then set ij as a permanent edge.

Rule 3. If ij, jk are permanent edges then set ik as a permanent edge.

Rule 4. If ij is a permanent edge and jk is a permanent non-edge then set ik as a permanent non-edge.

We remark that Rule 2 is also static and is equivalent to dealing with the S-vertices in Protti et al (2009) or the critical cliques in Guo (2009). Rules 3 and 4 can be applied either in a pre-processing step or at run time. In the latter case, for instance, they can be applied at each node of a branch-and-bound procedure, and the result depends on previous choices of variable values made in the current branch of execution.

In Section 10, the impact of these rules will be analyzed.

6 The Proposed GRASP Heuristic

GRASP (Resende and Ribeiro, 2003) is a multi-start metaheuristic for optimization problems, in which each iteration basically consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored until a local optimum is found during the local search phase. The best overall solution is stored as the result. In the remainder of this section, we describe in detail the phases of the proposed GRASP.

The complete GRASP pseudocode is illustrated in Algorithm 1. Initially, the input graph G and the maximum execution time t_{max} are taken as input in step 1. Next, a solution is generated and defined as the best solution by applying the construction procedure in step 2. In step 3, the control variable t_{start} is initialized. From lines 4 to 10, the GRASP iterations are performed. Note that the construction occurs in step 5, while the local search is carried out in step 6. If any improvement occurs, the best solution is updated in step 8. Finally, if the maximum time is reached the incumbent solution is returned in step 11.

Algorithm 1 GRASP pseudocode

```
1:  procedure GRASP($G, t_{max}$)
2:      $G^* \leftarrow construction(G)$
3:      $t_{start} \leftarrow time()$
4:      while $time() - time_{start} < time_{max}$ do
5:          $G' \leftarrow construction(G)$
6:          $G' \leftarrow LocalSearch(G')$
7:          if $s(G') < s(G^*)$ then
8:              $G^* \leftarrow G'$
9:          end if
10:     end while
11:     return $G^*$ as output
12: end procedure
```

6.1 Construction Phase

Given a graph G, the construction phase builds and evaluates, step-by-step, a feasible solution of CEP(G). This evaluation is given by the sum of costs of edges edited to transform G into this solution. As we will see, this is carried out in $O(n^2)$ time complexity. We remark that only feasible solutions are considered as the final product of the construction phase.

We have developed two greedy constructive heuristics: Relative Neighborhood (RN) and Vertex Agglomeration (VA). The aim of these algorithms is to generate distinct and relatively good solutions in limited computing time. Since the GRASP construction phase consists of generating a new solution at each iteration, a construction algorithm is chosen randomly or cyclically.

6.1.1 Relative Neighborhood

The idea behind the RN heuristic is to fix the initial vertex of each cluster, and then complete the cluster by choosing vertices that maximize the *relative neighborhood*. First, a Candidate List (CL) is built by sorting all vertices $i \in V(G)$ in decreasing order of $|N(i)|$. Next, the RN heuristic selects the first K vertices from CL to be cluster seeds, i.e., each selected vertex is the first element of a new cluster C_i. Let K_{best} be the number of clusters associated to the best current solution. At each iteration, the algorithm randomly selects a value of K between K_{min} and K_{max}, where $K_{min} = \max(K_{best} - \sqrt{n}, 1)$, $K_{max} = \min(K_{best} + \sqrt{n}, n)$, and $K = rand(K_{min}, K_{max})$. Finally a new vertex j from CL is randomly picked and added to one of the existing K clusters; the selected cluster C_i maximizes $RN(j, C_i) = cost^{+}(j, C_i) - cost^{-}(j, C_i)$. The process continues until CL becomes empty. At this moment, the K clusters guarantee the feasibility of the constructed solution, since all the edges connecting distinct clusters are removed and all non-edges inside a cluster are converted into edges. The time complexity of the RN heuristic is $O(n^2)$.

6.1.2 Vertex Agglomeration

The VA heuristic is a variant of the RN heuristic, with two main differences. The first one is the K cluster seeds are randomly picked in order to have more diversification. The second one is the remaining vertices are added according to the expression $VA(j, C_i) = cost^{+}(j, C_i)$ as an alternative to cluster construction. The time complexity of the VA heuristic is also $O(n^2)$.

6.2 Local Search Phase

After a solution is constructed, the local search phase is executed as an attempt to improve the initial solution. Three neighborhoods structures are proposed for the local search phase: Cluster Split (CS), Empty Cluster (EC), and Vertex Move (VM).

6.2.1 Cluster Split

In CS neighborhood, the cluster C with maximum value $\sum_{i \in C} cost^+(i, V(G) \setminus C)$ is split in order to improve overall solution quality. The splitting process consists of selecting, as cluster seeds, two vertices $i, j \in C$ that are non-neighbors in G such that $cost^+(i, C)$ and $cost^+(j, C)$ are the largest over C. The remaining vertices in $C \backslash \{i, j\}$ are allocated to the new clusters $\{i\}$ or $\{j\}$, according to the number of common neighbors. The time complexity to find the CS neighborhood is $O(n^2)$. In Fig. 2, the cluster C_a is split into two new clusters, C_b and C_c.

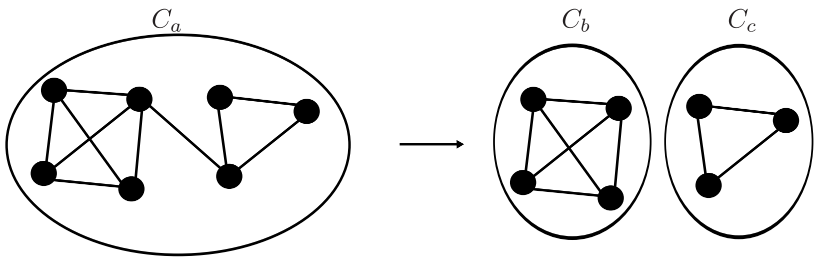


Fig. 2 Cluster Split Neighborhood.

6.2.2 Empty Cluster

The idea of the EC neighborhood is to find a cluster C in with there are too many original edges of G linking vertices in C to vertices in other clusters. In this case, a good strategy is to eliminate C by transferring its vertices to other clusters.

After identifying the candidate cluster C, which is the one with maximum value $\sum_{i \in C} cost^+(i, V(G) \setminus C)$, each vertex $i \in C$ is moved to another cluster C'; the choice is based on values $cost^+(i, C')$. This procedure terminates when $C = \emptyset$. The time complexity to find the EC neighborhood is $O(n^2)$. In Fig. 3, the cluster C_a is eliminated.

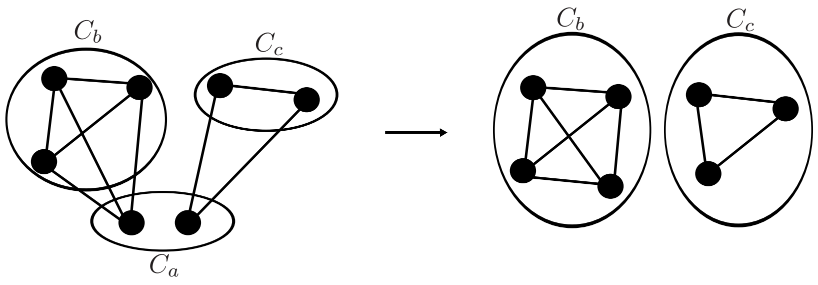


Fig. 3 Empty Cluster Neighborhood.

6.2.3 Vertex Move

The VM neighborhood tries to obtain good solutions by doing vertex moves, as shown in Fig. 4. Each move consists of removing a vertex from its original cluster and adding it to the cluster that maximally improves solution quality. Since it is computationally expensive to evaluate the new solution from scratch after each move (it takes $O(n^2)$), an alternative evaluation when moving a vertex i from cluster C to cluster C' is done by simply adding $cost^+(i,C) - cost^-(i,C) - cost^+(i,C') + cost^-(i,C')$ to the cost of the previous solution. This evaluation is carried out in $O(n)$ time. Since there are $|G'|$ clusters in solution G', moving one vertex can be performed in $O(n|G'|)$ time. Overall, the computation complexity of VM is $O(n^2|G'|)$.

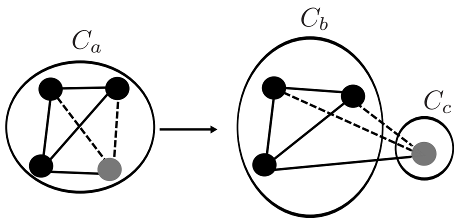


Fig. 4 Vertex Move Neighborhood.

6.2.4 A Variable Neighborhood Descent Procedure

Instead of using a simple search strategy, it is interesting to explore the search strategies of the Variable Neighborhood Descent (VND) procedure (Hansen and Mladenovic, 2003). The basic idea of VND is to systematically change the neighborhood, returning to the initial neighborhood when a better solution is found.

Algorithm 2 shows the pseudocode for the local search procedure. The algorithm starts by obtaining a solution G from the construction phase and the set of neighborhoods $\mathcal{N}$ (step 1). In step 2, the ordering of neighborhoods is randomized (Souza et al, 2011). In Penna et al (2013), computational experiments suggested that this approach is capable of finding better solutions when compared to those that adopt a deterministic order. Steps 4 to 12 perform the local search itself: a new solution G' is produced from the current neighborhood $\mathcal{N}_j$ in step 5. If solution G' is better than the incumbent solution G, the latter is updated and the neighborhood index j is restarted. Otherwise, j is incremented to allow the next neighborhood to be selected. This procedure ends when all the neighborhoods have been selected and no solution update is performed. The incumbent solution is returned in step 13.

7 The Proposed ILS Heuristic

ILS is a metaheuristic framework that integrates constructive, local search and perturbation procedures (Lourenço et al, 2003) . In the proposed ILS heuristic, initial

Algorithm 2 Local Search

```
1: procedure LOCAL SEARCH($G$,$\mathcal{N}$)
2:     randomize $\mathcal{N}$
3:     $j \leftarrow 1$
4:     while $j \leq j_{max}$ do
5:         $G' \leftarrow \mathcal{N}_j(G)$
6:         if $cost(G') < cost(G)$ then
7:             $j \leftarrow 1$
8:             $G \leftarrow G'$
9:         else
10:            $j \leftarrow j + 1$
11:        end if
12:    end while
13:    return $G$ as output
14: end procedure
```

solutions are generated using the constructive procedure described in Section 6.1, while local search is performed as explained in Section 6.2. Whenever a solution gets trapped in a local optimum, a perturbation is applied over this solution.

The pseudocode of the proposed ILS is presented by Algorithm 3. The algorithm gets as input a graph G, the maximum execution time, and the time interval to restart the search from a new initial solution. Firstly, a local optimum G^* is found (line 1). In lines 2 and 3, the best solution G_{best} and the control variables t_{start} and t' are initialized. Next, from lines 5 to 18, the ILS iterations are performed. Each iteration is composed by the following steps. A new solution is generated by randomly selecting a perturbation mechanism, and a local search is applied (line 6) and the resulting local optimum solution is stored in G^{**}. If the value is better than the value of G_{best}, then an update is performed in line 8. The acceptance criterion is applied over G^* to decide from which solution the search should continue. (lines 10 and 11). This criterion is greedy, i.e., G^* is only accepted if its value is better than the previous one. Finally, the method is restarted if it has not been updated within a fixed time period $t_{restart}$ (lines 14 and 15). The best solution generated by the algorithm is returned (line 19).

In the following subsections, we present the perturbation mechanisms proposed in this work. The worst-case time complexity of all of them is $O(n^2)$.

7.1 Random Cluster Change

Random Cluster Change (RCC) is a simple and straightforward perturbation. Given a solution G^*, RCC consists of selecting a vertex $i \in V(G^*)$ and transferring it to a distinct cluster, also chosen at random. The number of perturbation moves is nf_p, where $f_p \in [0, 1]$ is the perturbation rate (an input parameter of the algorithm). In order to calibrate the value for f_p, empirical tests were performed in a small subset of instances chosen at random. The test consisted in using several values of f_p for each one of the selected instances and then verifying which value yielded the best results.

Algorithm 3 ILS($G, t_{max}, t_{restart}$)

```
1:  $G^* \leftarrow LocalSearch(construct(G))$
2:  $G_{best} \leftarrow G^*$
3:  $t_{start} \leftarrow time()$
4:  $t' \leftarrow time()$
5:  while $(time() - t_{start}) < t_{max}$ do
6:      $G^{**} \leftarrow LocalSearch(perturb(G^*))$
7:      if $cost(G^{**}) < cost(G_{best})$ then
8:          $G_{best} \leftarrow G^{**}$
9:      end if
10:     if $cost(G^{**}) < cost(G^*)$ then
11:         $G^* \leftarrow G^{**}$
12:         $t' \leftarrow time()$
13:     end if
14:     if $(time() - t') \geq t_{restart}$ then
15:         $G^* \leftarrow LocalSearch(construct(G))$
16:         $t' \leftarrow time()$
17:     end if
18: end while
19: return $G_{best}$
```

In Fig. 5, vertex i was transferred from cluster C_a to cluster C_b.

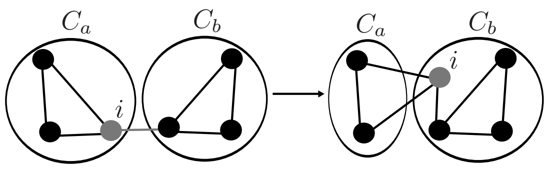


Fig. 5 Random Cluster Change Perturbation.

7.2 Cluster Split

Given a solution G^*, Cluster Split (CS) randomly selects a cluster C and partitions it into $|C|$ new clusters, each composed of a single vertex, as shown in Fig. 6. The main idea is to open the possibility of regrouping vertices of C in a different way, in order to build new clusters.

7.3 Cluster Creation

The Cluster Creation (CC) perturbation randomly selects a vertex $i \in V(G^*)$ and creates a new cluster C with it. Next, for every vertex $j \in V(G^*)\backslash\{i\}$, a random

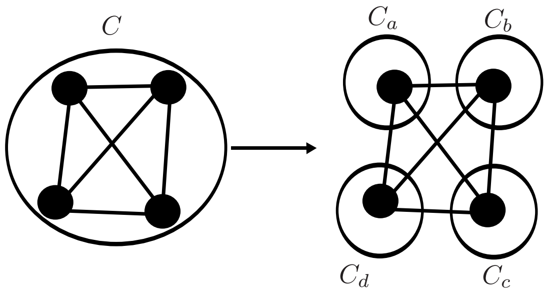


Fig. 6 Cluster Split Perturbation.

vertex k in the same cluster of j is chosen. If $cost(ij) > cost(jk)$ then k is transferred to C; if $cost(ij) < cost(jk)$ then k is kept in the same cluster; finally, if $cost(ij) = cost(jk)$ then k is transferred to C with a probability p, as shown in Fig. 7. The value of p was defined using the same methodology used to select the value of f_p in RCC.

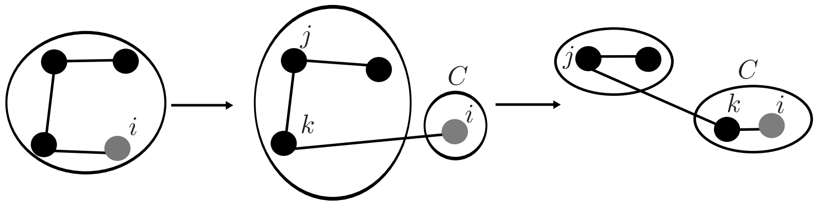


Fig. 7 Cluster Creation Perturbation.

8 A Refinement Method Based on Set Partitioning

In this section, we propose a method based on a two-phase strategy that works as follows. In the first phase, we generate a solution G^* of CEP(G) by applying an heuristic algorithm H (which is either the GRASP or the ILS heuristic, described in Sections 6 and 7) to input G. Consider all the feasible solutions $G_1, G_2, \ldots, G_k$ visited by H in order to obtain G^*. Assume that $V(G_j)$ is partitioned into clusters $C_j^1, \ldots, C_j^{r_j}$, and consider the collection $\mathcal{C}$ of all clusters generated in the first phase, that is, $\mathcal{C} = \cup_{j=1}^{k}\{C_j^1, \ldots, C_j^{r_j}\}$. We remark that $\mathcal{C}$ is not considered as a multiset (possible duplicates of a cluster C_j^h are discarded). Rewrite $\mathcal{C} = \{C_1, C_2, \ldots, C_\ell\}$. Note that each subset $C_j \in \mathcal{C}$ is a candidate to be a cluster in an optimal solution.

In the second phase, the idea is to select some suitable clusters from the pool of clusters $\mathcal{C}$ such that the selected clusters form a partition of $V(G)$. Let $\mathcal{C}_i \subseteq \mathcal{C}$ be the subset of clusters that contain vertex $i \in V(G)$. Define y_j as the binary variable associated with a cluster $C_j \in \mathcal{C}$, and let $c_j = cost(C_j)$, as defined in Section 2. We solve the following Set Partitioning (SP) formulation for CEP(G):

$$\text{minimize} \quad \sum_{j=1}^{\ell} c_j y_j \tag{4}$$

subject to

$$\sum_{j \in \mathcal{C}_i} y_j = 1 \qquad \forall i \in V(G) \tag{5}$$

$$y_j \in \{0,1\} \qquad 1 \leq j \leq \ell \tag{6}$$

The objective function (4) minimizes the sum of the costs by choosing the best combination of clusters. Constraints (5) state that every vertex $i \in V(G)$ must be exactly in a single cluster from the subset $\mathcal{C}_i$ (thus the selected cliques with $y_j = 1$ form a partition of $V(G)$). Constraints (6) define the domain of the decision variables.

Note that the objective function minimizes the number of edge editing operations over all possible solutions of CEP(G) that can be constructed from clusters in $\mathcal{C}$. The above ILP has always a solution, since $\mathcal{C}$ contains subcollections forming partitions.

The algorithm proposed above is called H+SP, which is the combination of the GRASP heuristic described in Section 6 or the ILS heuristic described in Section 7 with an exact method based on Set Partitioning (SP).

Algorithm 4 contains the pseudocode of algorithm H+SP. First, an empty pool of clusters is initialized (line 1). Next, a solution G^* is generated using heuristic H, which also fills the pool with the clusters (line 2). The SP model, given by (4)-(6), is created using the pool of clusters (line 3). The SP problem with G^* as a primal solution is then passed to a Mixed Integer Programming (MIP) solver (line 4) which, in turn, will find a solution G' at least as good as G^*. In case of improvement a local search is performed (lines 5-7).

Algorithm 4 H+SP(*MaxIter*, *MaxHTime*, *MaxSPTime*)

```
1: ClusterPool ← NULL
2: $G^*$ ← H(MaxIter, MaxHTime, ClusterPool)
3: SPModel ← CreateSPModel(ClusterPool)
4: $G'$ ← MIPSolver(SPModel, $G^*$, MaxSPTime)
5: if $f(G') < f(G^*)$ then
6:     $G^* \leftarrow LocalSearch(G')$
7: end if
8: return $G^*$
```

9 Generating Random Instances for the CEP

We discuss in this section how to generate random instances for CEP with various levels of difficulty. We follow the well-known random graph model proposed in Gilbert (1959). Denote by $G(n, p)$ a random graph with n vertices such that every possible edge occurs independently with probability p. To determine the 'difficulty' of an instance, we use a criterion based on number of edge editing operations necessary to achieve optimality (although other criteria, such as processing time, are also possible): for two instances G_1, G_2 *with the same number* n *of vertices*, we say that G_1 is *harder* than G_2 (or that G_2 is *easier* than G_1) if $opt(G_1) > opt(G_2)$.

The expected number of edges of $G(n,p)$ is $pn(n-1)/2$. The number p can also be viewed as the *density* (expected percentage of edges) of $G(n,p)$. Intuitively, using values of p close to 0 or 1 (related, respectively, to sparse or dense graphs) tend to generate easier instances. An algorithm for CEP($G(n,p)$) is expected to perform few edge additions in the former case, and few edge deletions in the latter. Such a behavior is experimentally tested below.

The value $opt(G(n,p))$ was calculated for several pairs (n,p). Figure 8 shows the results for some values of n. For a fixed n, each mark in the set $\{+, \times, *\}$ with coordinates (x,y) represents an experiment "set $p = x$, generate a random graph $G(n,p)$ and compute $y = opt(G(n,p))$". Note that, in general, harder instances are concentrated around $p = 0.6$. Figure 9 shows the corresponding percentages of deletions among the total number of editing operations.

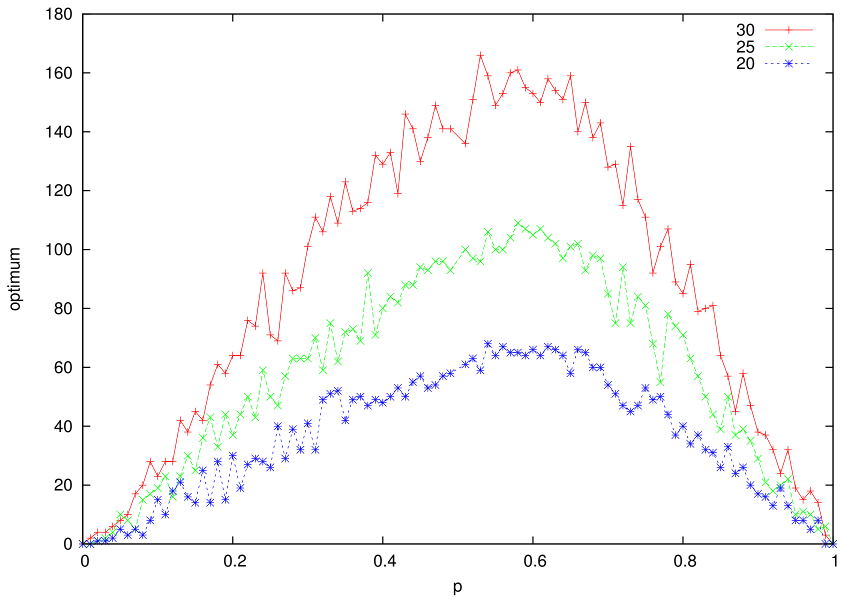


Fig. 8 Values $opt(G(n,p))$ for $n = 20, 25, 30$.

In order to interpret Figures 8 and 9, we resort to the theory of random graphs. In particular, we study the diameter of the random graph $G(n,p)$. Consider the property P: "no two vertices i,j of G satisfy $d_G(i,j) \geq 3$". Property P is clearly hereditary on subgraphs. Due to this fact, we can translate results from the Erdős-Rényi model of random graphs (Erdős and Rényi, 1959) to the Gilbert model. Denote by $[x]$ the

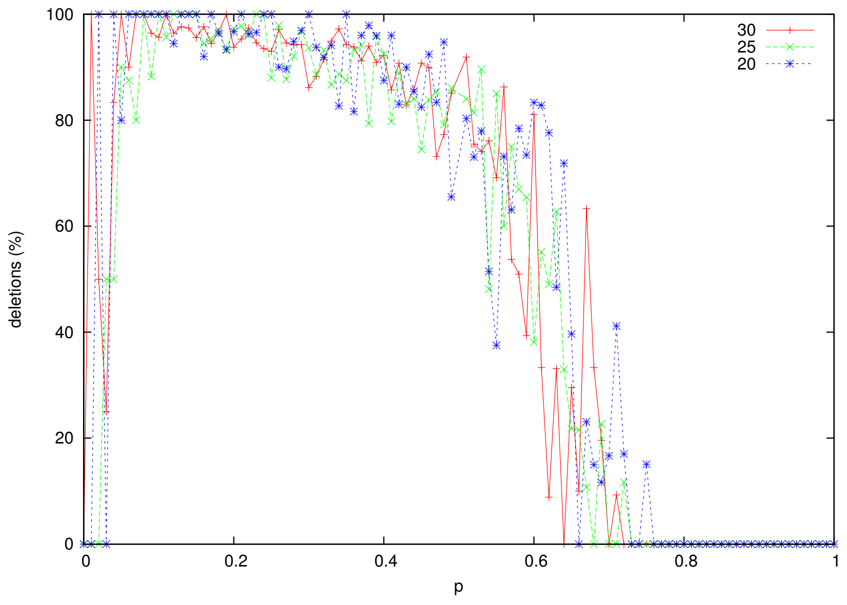


Fig. 9 Percentages of edge deletions among the total number of edge editing operations.

closest integer to x. For a graph G with n vertices and m edges, where $m = [pn(n-1)/2]$, let $\xi(n,m)$ be the expected percentage of pairs $(i,j) \in V(G) \times V(G)$, with $i < j$, such that $d_G(i,j) \geq 3$. By the proof of Lemma 1 in Klee and Larman (1981), p.623, $\xi(n,m) \leq (1-p^2)^{(n-3)/2}$. For instance, for $n = 25$ and $p = 0.4$, we have $\xi(n,m) \leq 5\%$. This shows that the probability of finding a pair of vertices at distance three or more in G is small when $p \geq 0.5$. Thus:

- For harder instances (those concentrated around $p = 0.6$, as shown in Figure 8), the most likely value for their diameter is two; this fact was already expected, due to Theorem 1. Optimal solutions are obtained by balanced numbers of edge deletions and additions (see Figure 9).
- For easier instances, Figure 9 shows two scenarios. For $p \leq 0.4$, edge deletions dominate edge additions, while for $p \geq 0.7$ the opposite situation occurs.

10 Experimental Results

In this section, the proposed algorithms are experimentally evaluated on artificial test problems. To the best of our knowledge, there are no public sites containing CEP instances. The instances used in our experiments were generated according to the method discussed in Section 9. Overall, there are 50 instances with 25 to 500 vertices and densities varying from 0.05 to 0.5.

All the tests have been run on a PC desktop equipped with a 6-core 3.33 GHz Intel Core I7 980X processor and 24 GBytes of RAM under Windows 7 x64 operating system. CPLEX 12 was used to solve SP problems to optimally, by means of the formulation presented in Section 4. All the algorithms were code in C++ and compiled with MinGW-x64.

In order to evaluate the convergence of the heuristic algorithms, preliminary tests were performed in order to define a target solution value $tgt(G)$ for each instance G, as follows. If the solver was capable of finding $opt(G)$ then $tgt(G) = opt(G)$, otherwise the optimal solution is unknown and $tgt(G)$ was set as the best solution value found in 10 runs of the GRASP heuristic; the idea in this case is to set an "average difficulty" target with a view of balancing convergence rate results of the heuristic algorithms.

10.1 Evaluation of the Algorithms

Preliminary experiments revealed that *Relative Neighborhood* appears to be more efficient than *Vertex Agglomeration*, although, in some instances, the latter is capable of generating a better initial solution. Nevertheless, from our tests, we observed that a high quality initial solution does not necessarily imply in a good final solution. It is actually more interesting to start with diversified solutions, than to put efforts in devising constructive procedures aiming at building high quality initial solutions. In view of this, we decided to include both methods in our algorithm.

The same behavior occurs for the local search and perturbation mechanisms. Their individual performance appear to be dependent on the instance characteristics. However, when put together, they seem to perform much better. More specifically, if we run different versions of the algorithm using the same CPU time limit, where in each of them we consider only one neighborhood structure, the quality of the solutions generated by these versions will not be as good as those generated by the version that makes use of all neighborhood structures.

Our first experiment consisted of running the CPLEX solver, using the ILP described in Section 4, for each instance. A time limit of three hours was imposed for each run. Table 1 shows the number of instances solved to optimally, for each group size. Column *n* denotes the size of each group of instances, column *total* correspond to the number of instances of each size, and column *opt* indicates the number of instances solved to optimally. CPLEX was only capable of finding the optimal solutions of 28% of the instances. Also, no instance with more than 100 vertices was solved.

Table 1 Number of instances optimally solved.

n	*total*	*opt*
25	10	10
50	10	3
100	10	1
200	10	0
500	10	0
1000	10	0
	50	14

Table 2 summarizes the results obtained by GRASP and ILS for small instances (graphs with at most 200 vertices). The first column corresponds to the size of each group of instances, column *total* specifies the number of instances of each size, and the third column indicates the time limit in seconds used as a stop criterion. Columns labeled *conv%* show the percentage of times the target $tgt(G)$ was found by the algorithm, where 10 executions were performed for each instance G. Finally, columns labeled *time* denote, in each row, the average runtime in seconds for all the runs associated with that row.

Table 2 Experimental results for small instances.

			GRASP		ILS	
n	*total*	t_{max}	*conv %*	*time* (s)	*conv %*	*time* (s)
25	10	0.63	100.00	0.00	100.00	0.01
50	10	2.50	60.00	0.72	94.00	0.27
100	10	10.00	42.00	7.59	97.00	0.48
200	10	40.00	23.00	7.59	100.00	0.74
	40		56.25	3.97	97.75	0.38

From Table 2 it can be observed that, given a time limit t_{max}, the convergence rate of GRASP decreases as n increases, while ILS maintains a good performance and regular behavior for all values of n, with small runtime.

Table 3 shows the results of experiments using a time limit as the stopping criterion. From left to right, the columns show: the size of the instances, the number of instances of each size, and the computational time of each algorithm according to the size of the instances. The columns named gap$_b$% show the average percentage gap with respect to the best solution found for each instance, while the columns named gap$_a$% show the average percentage gap considering all solutions found for each instance. The gaps are calculated with respect to the optimal solution or with respect to the best known solution (when the optimal solution is unknown). For every instance with known optimal value, both GRASP and ILS found the optimal value.

Table 3 The gap between GRASP and ILS in small instances.

			GRASP		ILS	
n	*total*	t	*gap$_b$ %*	*gap$_a$ %*	*gap$_b$ %*	*gap$_a$ %*
25	10	0.31	0.00	0.00	0.00	0.00
50	10	1.25	0.00	0.23	0.00	0.06
100	10	5.00	0.99	1.30	0.00	0.18
200	10	20.00	1.13	1.34	0.00	0.21
	40		0.53	0.71	0.00	0.11

Table 4 presents detailed results for $n = 500$. Each row in this table corresponds to a single instance. The first column shows the expected density d of the instance (see Section 9). Remaining columns have a similar meaning as in Table 2. However, in this case, the values in each row are calculated over 10 runs on the same instance. A time limit of 250 seconds was imposed for all the runs.

Table 4 Convergence rate using $t_{max} = 250$ s for instances with $n = 500$.

d	GRASP	ILS	GRASP+SP	ILS+SP
0.05	0.00	60.00	100.00	100.00
0.10	0.00	100.00	100.00	100.00
0.15	10.00	100.00	20.00	100.00
0.20	0.00	100.00	10.00	100.00
0.25	30.00	100.00	30.00	100.00
0.30	20.00	100.00	50.00	100.00
0.35	10.00	100.00	10.00	100.00
0.40	10.00	100.00	40.00	100.00
0.45	0.00	100.00	20.00	100.00
0.50	30.00	100.00	60.00	100.00
	11.00	96.00	44.00	100.00

By observing the results described in Table 4, one can verify that when embedding the SP module into GRASP, the convergence rate increases by four times. On the other hand, when incorporating the SP module into ILS, where the convergence rate was 100% for all instances.

Table 5 presents detailed results for $n = 1000$. As in Table 4, the values on each row are computed over 10 runs on the same instance. However, in this case, a time limit of 400 seconds was imposed for all runs.

Table 5 Convergence rate using $t_{max} = 400$ s for instances with $n = 1000$.

	GRASP	ILS	GRASP+SP	ILS+SP
0.05	50.00	80.00	50.00	100.00
0.10	80.00	100.00	100.00	100.00
0.15	80.00	60.00	100.00	100.00
0.20	60.00	60.00	100.00	100.00
0.25	90.00	60.00	100.00	100.00
0.30	60.00	30.00	100.00	100.00
0.35	50.00	60.00	100.00	100.00
0.40	70.00	70.00	100.00	100.00
0.45	80.00	70.00	100.00	100.00
0.50	30.00	60.00	100.00	100.00
	65.00	65.00	95.00	100.00

The results described in Table 5 show that both GRASP and ILS have the same average convergence rate. As in Table 4, the convergence rate also increases when embedding the SP module. Again, when incorporating the SP module into ILS, the convergence rate was 100% for all instances.

10.2 Impact of Reduction Rules

In this subsection, we first analyze the effectiveness of reduction rules in finding *permanent edges/non-edges* (see Section 5). Table 6 illustrates the percentages ("fixation rates") of edges and non-edges that have been set as *permanent* after applying the rules. The first column indicates the size of each group of instances; second and third columns show fixation rates for edges (Rules 2 and 3) and non-edges (Rules 1 and 4), respectively; while fourth and fifth columns show the fixation rate of Rule 1 and the expected rate giving by the $\xi(n, m)$ value (see Section 9), respectively. The present average values were computed over 10 instances of the same size. Since Rules 2 and 3 (see Section 5) are only preprocessing procedures for fixing permanent edges, we note that its effectiveness is not significant for the instances tested. In addition, for permanent non-edges, fixation rates are substantial for smaller graphs but such rates rapidly decrease as the instances get larger. Regarding Rule 1, we can verify that this rule is responsible for the most of non-edges fixing and close to the theoretical expected value $\xi(n, m)$.

Table 6 Fixation rates in terms of n.

n	*edges %*	*non-edges %*	*Rule 1 %*	$\xi(n, m)$
25	1.67	32.47	21.92	32.16
50	0.00	19.55	15.04	20.65
100	0.00	12.28	10.22	12.93
200	0.00	7.73	7.10	7.60
500	0.00	3.08	2.86	2.94
1000	0.00	0.82	0.70	0.82
	0.28	**12.66**	**9.64**	**12.85**

As pointed out in Section 9, graph density has a significant influence on fixing permanent edges/non-edges. Table 7 presents the fixation rates with respect to the graph density. The first column shows instance densities, while second to fifth columns are similar to Table 6, where these values were computed over 6 instances with the same density with $n \in \{25, 50, 100, 200, 500, 1000\}$. These results confirm that the fixation rates tend to increase considerably in sparser graphs, where Rule 1 obtains results close to the theoretical expected value $\xi(n, m)$.

In addition to reduction rate analysis, we first investigated how the CPLEX solver performs on raw instances and then over reduced instances. The objective is to verifiy if reduction rules affect runtime and solution quality. To evaluate the effect of reduction rules in the runtime, we have computed the improvement on the average time as follows: $time_{imprv} = (time_{raw} - time_{red})/time_{raw}$, where $time_{raw}$ and $time_{red}$ are the average runtimes for raw and reduced instances, respectively. To evaluate the effect of reduction rules over the solution quality, we have computed the gap difference $\Delta gap = gap_{raw} - gap_{red}$, where gap_{raw} (gap_{red}) is the average gap obtained by the solver on raw instances (reduced instances, respectively). Table 8 depicts the CPLEX results, where the first column represents the instance sizes, while second and third columns show $time_{imprv}$ and Δgap values.

Table 7 Fixation rates in terms of d.

d	*edges %*	*non-edges %*	*Rule 1 %*	$\xi(n,m)$
0.05	4.17	80.52	56.33	60.02
0.10	0.00	45.36	26.93	32.45
0.15	0.00	27.52	9.07	17.87
0.20	0.00	13.77	3.02	9.56
0.25	0.00	8.78	0.45	4.86
0.30	0.00	1.95	0.17	2.29
0.35	0.00	1.37	0.17	0.98
0.40	0.00	0.81	0.17	0.36
0.45	0.00	0.00	0.10	0.12
0.50	0.00	0.00	0.00	0.03
	0.42	**18.01**	**9.64**	**12.85**

Table 8 Time and solution quality improvements in terms of n.

n	$time_{imprv}$ %	Δgap %
25	3.67	0.00
50	11.47	0.26
100	9.53	16.35
200	-	147.05
	8.22	**40.92**

Time improvement results in Table 8 do not draw a clear pattern since most instances could not be solved within the time limit, yielding an unrealistic time difference between raw and reduced instances. However, we remark that: (a) for a few instances, the runtime decreased 95% after using reduction rules; (b) when considering those instances solved to optimally using both raw data and reduced data, time improvement was, on average, 41% – in these cases, interestingly, CPLEX speed up using reduction rules increases with the instance size. Therefore, gains are highly meaningful, which strongly justifies the use of the reduction rules studied.

10.3 Comparison with other Algorithms

In this section, we compare the ILS-SP with the Transitivity Clustering method (TC) (Wittkop et al, 2007). The TC method combines the heuristics Force-Based Cluster Editing (FORCE) (Wittkop et al, 2007) and Cluster Affinity Search Technique (CAST) (Ben-Dor et al, 1999) with the exact algorithm Fixed-Parameter (FP) (Böcker et al, 2008) to solve the CEP. More specifically, TC seeks to balance the computational time and solution quality by combining the algorithms according to certain criteria. For example, exact methods are not employed on large instances, but they are used in small instances.

The authors of the TC method compared their approach against the best known methods available in the literature and show that, in general, TC obtains high quality solutions with low computing time (Wittkop et al, 2010), when compared to other heuristic algorithms proposed in the literature. Because of this, and also due to the fact that TC gathers the most succesful methods from the literature, we only compare our results with those obtained by the TC algorithm.

We downloaded the TC software in Baumbach et al (2014) and conducted the tests on the machine described at the beginning of Section 10. The software does not allow the user to set a time limit. To perform a fair comparison we run the TC software 10 times on the selected instances and store the results. Then we use the average computational times of TC as an input limit time for the ILS-SP. Moreover, we also executed our algorithm 10 times.

Table 9 illustrates the comparison between ILS-SP and TC. From left to right, the columns show: the size of each instance, the average graph density, the computational time used by the algorithms, and the gaps of the best solutions found by each algorithm. From Table 9 we can observe that ILS-SP finds better solutions in all instances but one. Moreover, the average gap of ILS-SP was 0.01% against 0.11% of TC.

Table 9 Comparison between TC and ILS-SP

			TC	ILS-SP
n	d	t	*gap %*	*gap %*
1000	0.05	6.6	0.19	0.00
1000	0.10	6.1	0.02	0.00
1000	0.15	6.7	0.00	0.06
1000	0.20	19.6	0.01	0.00
1000	0.25	30.3	0.04	0.00
1000	0.30	12.1	0.02	0.00
1000	0.35	11.7	0.08	0.00
1000	0.40	11.6	0.13	0.00
1000	0.45	10.2	0.22	0.00
1000	0.50	19.3	0.36	0.00
			0.11	0.01

10.4 Empirical Probability

In this section we evaluate the algorithms GRASP and ILS using the methodology introduced by Aiex et al (2003), which consists of analyzing the results of each algorithm on certain instances empirically. The idea is to set a target value for the solution of the problem and store the time spent by the algorithm to find that target. Finally, easy and hard targets may be chosen to evaluate the behavior of each algorithm.

For an instance with size $n = 500$, three targets with different degrees of difficulty were defined. The first target, considered easy, was defined as the average cost of the solutions generated by GRASP during preliminary tests. The second target, assumed to have medium difficulty, was selected as the cost of the best solution found by GRASP, also according to preliminary tests. Finally, the third target, considered hard, was defined as the cost of the best known solution. For each instance and for each target, the algorithms were executed 100 times, and the computational time limit was 100 seconds for the easy target; 250 seconds for the medium target; and 1000 seconds for the hard one.

Figures 10, 11, and 12 show the probability distributions of the results obtained for the instance with easy, medium, and hard targets respectively. The figures show the computational time as the abscissa in logarithmic scale and the probability p_i as the ordinate. The results demonstrate that ILS converges more quickly than GRASP in all cases.

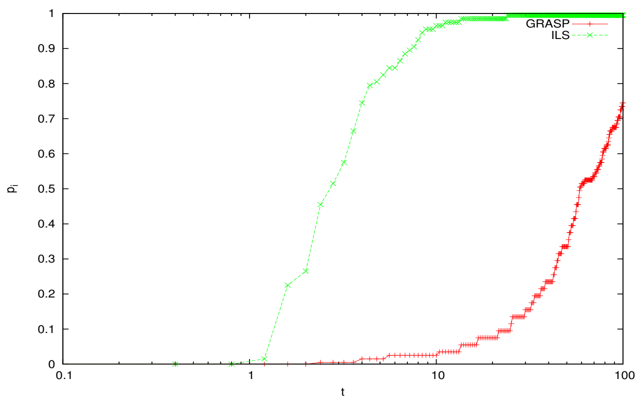


Fig. 10 Empirical probability: easy target.

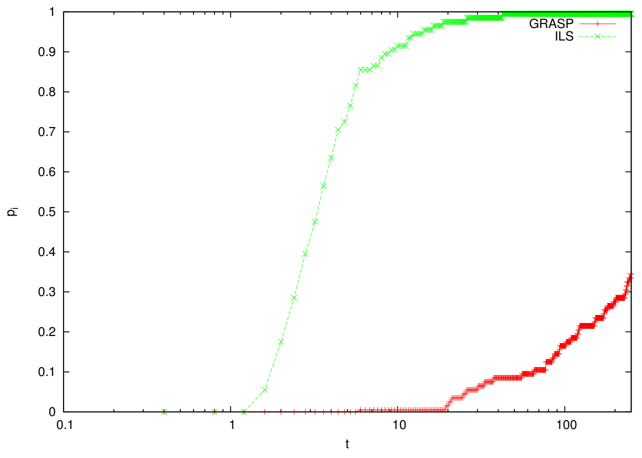


Fig. 11 Empirical probability: medium target.

11 Conclusions

In this paper, several contributions for the Cluster Editing Problem have been made. The most important ones include theoretical and empirical results about data reduc-

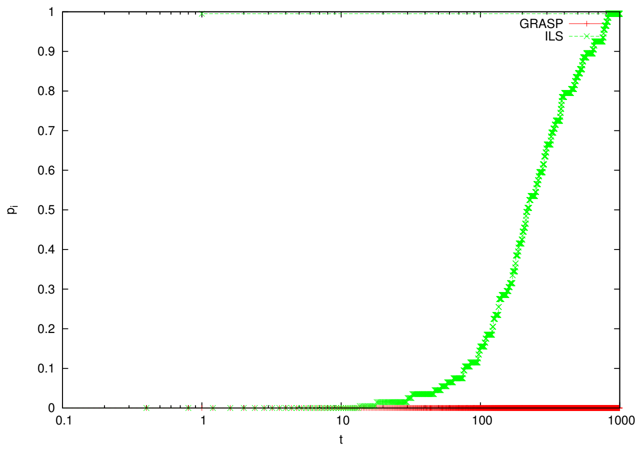


Fig. 12 Empirical probability: hard target.

tion and instance generation. Moreover, we propose fast algorithms that solve large instances in reasonable runtimes.

To the best of our knowledge, this is the first time that major efforts were made to study, generate and solve hard instances for the problem. In fact, we characterize a class of hard instances by empirical evidences, showing that the number of edge editing operations is greater when the graph has density about 0.6, and decreases when the graph becomes sparser or denser.

Two heuristics have been proposed based on the GRASP and ILS metaheuristics, as well as two new greedy constructive heuristics and three new neighborhoods. We have also developed three perturbation algorithms for an ILS-based heuristic.

In order to strengthen the results obtained by the proposed heuristic algorithms, an exact module based on Set Partitioning has been integrated to them, resulting in a very robust tool in terms of solution quality.

Future work includes searching for new valid inequalities for the formulation described in Section 4, in order to optimally solve more and larger instances and thus create a wider basis for the evaluation of the heuristics. Other interesting research direction is to develop a theoretical analysis of the proposed heuristic algorithms, e.g. study their approximation ratio, specially for graphs of diameter two (since the CEP is NP-hard even for such graphs).

References

Aiex RM, Binato S, Resende MGC (2003) Parallel grasp with path-relinking for job shop scheduling. Parallel Computing 29:393–430

Bastos LO (2012) Novos algoritmos e resultados teóricos para o problema de particionamento de grafos por edição de arestas. PhD thesis, Universidade Federal Fluminense, in Portuguese

Baumbach J, Emig D, Kleinbolting N, Lange S, Rahmann S, Wittkop T (2014) TransClust. URL http://transclust.mmci.uni-saarland.de/main_page/index.php, (accessed January 28, 2014)

Ben-Dor A, Shamir R, Yakhini Z (1999) Clustering gene expression patterns. J of Comput Biol 6(3/4):281–297

Böcker S, Briesemeister B, A QB, Truss A (2008) Going weighted: Parameterized algorithms for cluster editing. Comb Optim and Appl 5165:1–12

Böcker S, Briesemeister S, Klau G (2009) Exact algorithms for cluster editing: Evaluation and experiments. Algorithmica pp 1–19

Charikar M, Guruswami V, Wirth A (2005) Clustering with qualitative information. J of Comput and Syst Sci 71:360–383

Dehne F, Langston MA, Luo X, Pitre S, Shaw P, Zhang Y (2006) The cluster editing problem: Implementations and experiments. Lect Notes in Comput Sci 4169:13–24

Erdős P, Rényi A (1959) On random graphs i. Publicationes Mathematicae 6:290–297

Gilbert EN (1959) Random graphs. Annals of Math Stat 30:1141–1144

Gramm J, Guo J, Hüffner F, Niedermeier R (2005) Graph-modeled data clustering: Exact algorithms for clique generation. Theor Comput Sys 38:373–392

Grötschel M, Wakabayashi Y (1989) A cutting plane algorithm for a clustering problem. Math Program 45(1):59–96

Guo J (2009) A more effective linear kernelization for cluster editing. Theor Comput Sci 410:718–726

Hansen P, Mladenovic N (2003) Variable neighborhood search. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics, Kluwer Acad. Publ., chap 6, pp 145–183

Hartuv E, Schmitt AO, Lange J, Meier-Ewert S, Lehrach H, Shamir R (2000) An algorithm for clustering cdna fingerprints. Genomics 66(3):249–256

Jain AK, Murty MN, Flynn PJ (1999) Data clustering: A review. ACM Comput Surv 31(3):264–323

Klee V, Larman D (1981) Diameters of random graphs. Can J of Math 33(3):618–640

Lourenço HR, Martin OC, Stützle T (2003) Iterated local search. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics, Kluwer Acad. Publ., chap 11, pp 321–353

Milosavljevic A, Strezoska Z, Zeremski M, Grujic D, Paunesku T, Crkvenjakov R (1995) Clone clustering by hybridization. Genomics 27(1):83–89

Penna PHV, Subramanian A, Ochi LS (2013) An iterated local search heuristic for the heterogeneous fleet vehiclerouting problem. J of Heuristics 19(2):201–232

Protti F, Silva MD, Szwarcfiter J (2009) Applying modular decomposition to parameterized cluster editing problems. Theory of Comput Syst 44:91–104

Rahmann S, Wittkop T, Baumbach J, Martin M, Truss A, Böcker S (2007) Exact and heuristic algorithms for weighted cluster editing. In: Markstein P, Xu Y (eds) Comput. Syst. Bioinforma.: CSB 2007 Conf. Proc., Imp. Coll. Press, 57 Shelton Street, Covent Garden, London WC2H 9HE, vol 6, pp 391–400

Resende M, Ribeiro C (2003) Greedy randomized adaptive search procedures, Kluwer Acad. Publ., chap 8, pp 219–249

Sen Gupta A, Palit A (1979) On clique generation using boolean equations. In: Proc. of the IEEE, The IEEE, Inc., 345 East 47 Street, New York. NY 10017., vol 67, pp 178–180

Shamir R, Sharan R, Tsur D (2004) Cluster graph modification problems. Discret Appl Math 144:173–182

Sharan R, Maron-Katz A, Shamir R (2003) Click and expander: a system for clustering and visualizing gene expression data. Bioinforma 19(14):1787–1799

Souza MJF, Mine MT, de Silva MSA, Ochi LS, Subramanian A (2011) A hybrid heuristic, based on iterated local search and genius, for the vehicle routing problem with simultaneous pickup and delivery. Int J of Logist Syst and Manag 10(2):142–157

Tatusov R, Fedorova N, Jackson J, Jacobs A, Kiryutin B, Koonin E, Krylov D, Mazumder R, Mekhedov S, Nikolskaya A, Rao BS, Smirnov S, Sverdlov A, Vasudevan S, Wolf Y, Yin J, Natale D (2003) The cog database: an updated version includes eukaryotes. BMC Bioinforma 4(1):41

Wittkop T, Baumbach J, Lobo FP, Rahmann S (2007) Large scale clustering of protein sequences with FORCE - A layout based heuristic for weighted cluster editing. BMC Bioinforma 8:396

Wittkop T, Emig D, Lange SJ, Rahmann S, Albrecht M, Morris JH, Böcker S, Stoye J, Baumbach J (2010) Partitioning biological data with transitivity clustering. Nature Methods 7(6):419–420