PACE Solver Description: ADE-Solver*

Alexander Bille
Computer Science, Philipps Universität Marburg, Germany

Dominik Brandenstein
Computer Science, Philipps Universität Marburg, Germany

Emanuel Herrendorf
Computer Science, Philipps Universität Marburg, Germany

Abstract

This document describes our exact solver "ADE" for the unweighted cluster editing problem submitted to the PACE 2021 competition. The solver's core consists of an FPT-algorithm using a branch and bound strategy in conjunction with several data reduction rules.

2012 ACM Subject Classification Theory of computation → Branch-and-bound; Mathematics of computing → Graph algorithms

Keywords and phrases Unweighted Cluster Editing

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.28

Supplementary Material
Software (Source Code Archive): `https://doi.org/10.5281/zenodo.4889012`
Software (Source Code Repository): `https://github.com/EmanuelHerrendorf/pace-2021`
Software (Source Code Archive Version 2): `https://doi.org/10.5281/zenodo.4889825`
Software (Source Code Repository Version 2): `https://github.com/Araxon/pace-2021-v2/`
Software (Source Code Archive Version 3): `https://doi.org/10.5281/zenodo.4889897`
Software (Source Code Repository Version 3): `https://github.com/Araxon/pace-2021-v3/`

Acknowledgements We want to thank Christian Komusiewicz and Frank Sommer for being our mentors.

1 Introduction

The PACE 2021[1] competition features the Cluster Editing problem. The task of Cluster Editing is to find a minimum size set of edges to add or delete such that the resulting graph is a cluster graph, that is, every connected component is a clique.

Our solver is implemented in Java and its source code can be found at `https://doi.org/10.5281/zenodo.4889012` or via GitHub `https://github.com/EmanuelHerrendorf/pace-2021` . It expects the input graph via the standard input and outputs on the standard output one minimum set of edges to modify to obtain a cluster graph. To avoid stack overflow issues the stack size for the JVM should be increased using the `-Xss` parameter. Our solver is an FPT-algorithm based on a branch and bound strategy. The refined search tree for unweighted cluster editing introduced by Gramm et al. [5] is used as the branching strategy. It guarantees a search tree size of $O(2.27^k)$ where k refers to the minimum number of edge modifications needed to turn the input graph into a cluster graph. During branching edges

* This is a brief description of one of the highest ranked solvers of PACE Challenge 2021. It has been made public for the benefit of the community and was selected based on the ranking. PACE encourages publication of work building on the ideas presented in this description in peer-reviewed venues.

[1] `https://pacechallenge.org/2021`

can be marked as permanent and non-edges as forbidden indicating that these edges need not be modified to obtain an optimal solution.

2 Finding an Optimal Solution

In order to find an optimal solution for Cluster Editing one has to find a valid solution and prove the non-existence of another solution with fewer edge modifications. This can be done for each connected component of the input graph independently, described for example by Böcker et al. [2]. We solve the decision variant of Cluster Editing iteratively asking whether a solution can be found with at most k edge modifications. To arrive at an optimal solution as quickly as possible we try to choose a "good" initial value for k before starting our search tree and then go from there depending on the results we get while traversing it.

For setting the initial value of k we compute a lower and upper bound on the initial graph. These are then heuristically weighted (10% lower bound and 90% upper bound) in order to arrive at an initial value between both of them (version 1 and 3 start with the upper bound decremented by one instead). Also we keep the lower and upper bound and the heuristic solution saved for later use. We then begin traversing the search tree.

If we do not find a solution our current cost limit k is too low so we heuristically increase it by three at a time in order to avoid too many full traversals of unsuccessful search trees and then start traversing the search tree all over. If we have reached the point where we previously already have found a valid solution of size $k + 1$ (either the initial upper bound or another solution found during branching), we simply output this solution. Otherwise we update the lower bound to $k + 1$.

If we, however, do find a solution we have to consider two different cases in order to ensure it is indeed an optimal one:

1. The solution's size matches the current lower bound. Then we can immediately output the solution.
2. The solution is at least one larger than the current lower bound. Then there could still be a valid solution with a size smaller than our current solution. We know, however, that the part of the current search tree which we did already traverse cannot contain it. Otherwise we would have found it earlier instead of the current solution. So we simply continue the traversal of the current search tree with k decremented by one.

3 Branching

Before the branching data reduction rules are applied. After the data reduction, the lower bound is computed to check whether the current branch cannot be solved without exceeding the edge modification limit k. The search tree introduced by Gramm et al. [5] is based on branching on P_3s until no P_3 exists. During the branching a data structure containing all P_3s is maintained and updated according to the modified edges. Furthermore we maintain a conflict graph over all P_3s in which an edge represents two P_3s sharing one non-permanent edge or the non-forbidden non-edge. We define an ordering in which the P_3s are selected by the branching with respect to the following properties of a P_3. Primarily the P_3 with the higher number of permanent or forbidden edges is selected. The following properties will be used to break ties (all in descending order):

1. the number of neighbours in the previously mentioned P_3 conflict graph
2. the degree sum of the three vertices of the P_3

3. the sum of the common neighbours of non-permanent edges and non-common neighbours of non-forbidden non-edges
4. the maximum degree of the vertices of the P_3

If a tie still occurs after these properties have been taken into account, a random P_3 is selected.

4 Lower Bound

The lower bound used for restricting the size of the search tree is the size of a maximal conflict-disjoint P_3 packing which has been suggested by Hartung and Hoos [6]. We follow the incremental approach of Gottesbüren et al. [4] where an initial greedy packing is computed and further improved by local search. For the greedy step the P_3s are taken into account using the same ordering as defined for the branching except that the properties are used in ascending order. The local search tries to exchange one P_3 contained in the packing with at least two P_3s not being part of the packing until no such exchanges can be found. Before the lower bound is updated again, one-to-one exchanges are performed to replace P_3s in the packing with P_3s that are smaller in terms of the ordering.

5 Upper Bound

An upper bound is used in combination with the lower bound to select the k value to start the search tree with. The upper bound used for the parameter k is the size of a heuristically computed solution. This heuristic consists of two steps.

First, the graph is split into clusters using a greedy approach. The closed neighbourhood of a vertex with lowest modification cost is made a clique and removed from the graph until the graph is empty. These costs are the sum of the closed neighbourhood's cut weight and deficiency defined by Cao and Chen [3]. Furthermore, we extended this process such that a vertex from the second neighbourhood of vertex v will be inserted into the clique of v as well if they have at least as many common neighbours as non-common neighbours.

In the second step, these clusters are modified using local search. Three operations being variants of the cluster operations described by Bastos et al. [1] are performed exhaustively until the solution size cannot be decreased anymore. The first operation consists of removing a vertex from a cluster to create its own cluster. The second operation tries to move a vertex into another cluster. The last one merges two clusters. To increase the search area the last two operations are also applied if they leave the solution size unchanged and the total number of such applications does not exceed a threshold.

6 Data Reduction Rules

To reduce the size of the search tree, a set of data reduction rules is applied. We use the $(k+1)$-rule introduced by Gramm et al. [5] and improved by Hartung and Hoos [6]. This rule is applied on the whole graph every fourth search tree node and supplemented by a lighter variant local to recent edge edits and used in the other search tree nodes. If the rule was successfully applied on the whole graph, version 3 afterwards tries to improve the lower bound and then additionally applies the light variant. The following reduction rules are applied in every search tree node. We apply both P_3-rules introduced by van Bevern et al. [7]. Two other reduction rules suggested by Gramm et al. [5] are considered. First, the third (non-)edge of a vertex triple can be directly edited or marked as not editable if the other

two (non-)edges are already not editable and this triple otherwise would form a permanent P_3. The second rule deletes an edge of a P_3 if it is only connected to the rest of the graph through its middle node. Finally, we use a rule of Bastos et al. [1] which sets non-edges $\{u, v\}$ to forbidden if u and v do not have common neighbours.

References

1 Lucas Bastos, Luiz Satoru Ochi, Fábio Protti, Anand Subramanian, Ivan César Martins, and Rian Gabriel S Pinheiro. Efficient algorithms for cluster editing. *Journal of Combinatorial Optimization*, 31(1):347–371, 2016.

2 Sebastian Böcker, Sebastian Briesemeister, and Gunnar W Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.

3 Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. *Algorithmica*, 64(1):152–169, 2012.

4 Lars Gottesbüren, Michael Hamann, Philipp Schoch, Ben Strasser, Dorothea Wagner, and Sven Zühlsdorf. Engineering Exact Quasi-Threshold Editing. In Simone Faro and Domenico Cantone, editors, *18th International Symposium on Experimental Algorithms (SEA 2020)*, volume 160 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

5 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.

6 Sepp Hartung and Holger H. Hoos. Programming by optimisation meets parameterised algorithmics: A case study for cluster editing. In *Learning and Intelligent Optimization: 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*, volume 8994, page 43. Springer, 2015.

7 René van Bevern, Vincent Froese, and Christian Komusiewicz. Parameterizing edge modification problems above lower bounds. *Theory of Computing Systems*, 62(3):739–770, 2018.