PACE Solver Description: KaPoCE: A Heuristic Cluster Editing Algorithm*

Thomas Bläsius
Karlsruhe Institute of Technology, Germany

Philipp Fischbeck
Hasso Plattner Institute, Potsdam, Germany

Lars Gottesbüren
Karlsruhe Institute of Technology, Germany

Michael Hamann
Karlsruhe Institute of Technology, Germany

Tobias Heuer
Karlsruhe Institute of Technology, Germany

Jonas Spinner
Karlsruhe Institute of Technology, Germany

Christopher Weyand
Karlsruhe Institute of Technology, Germany

Marcus Wilhelm
Karlsruhe Institute of Technology, Germany

Abstract

The cluster editing problem is to transform an input graph into a cluster graph by performing a minimum number of edge editing operations. A cluster graph is a graph where each connected component is a clique. An edit operation can be either adding a new edge or removing an existing edge. In this write-up we outline the core techniques used in the heuristic cluster editing algorithm of the **Ka**rlsruhe and **Po**tsdam **C**luster **E**diting (`KaPoCE`) framework [2] (contains also an exact solver), submitted to the heuristic track of the 2021 PACE challenge.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases cluster editing, local search, variable neighborhood search

Digital Object Identifier 10.4230/LIPIcs.IPEC.2021.31

Supplementary Material *Software (Source Code)*: `https://doi.org/10.5281/zenodo.4892524`
Software (Source Code): `https://github.com/kittobi1992/cluster_editing`

1 Preliminaries

Let $G = (V, E)$ be a simple and undirected graph. $I(u) := \{v \mid (u, v) \in E\}$ denotes the set of all adjacent vertices of a vertex $u \in V$. A clustering $\mathcal{C} := \{C_1, \dots, C_l\}$ is a partition of the vertex set V into l non-empty and disjoint sets. A set $C \in \mathcal{C}$ is also called a *cluster*. We call a cluster $C \in \mathcal{C}$ a *singleton cluster*, if $|C| = 1$. *Isolating* a vertex u or moving it into its *own* cluster means to assign it to a new singleton cluster $C = \{u\}$. A vertex u is adjacent to a cluster $C \in \mathcal{C}$, if there exists a vertex $v \in I(u)$ with $v \in C$. We can transform G into a cluster graph where each cluster of a clustering $\mathcal{C}$ forms a clique by adding all edges of $E^+ := \{(u, v) \notin E \mid \exists C \in \mathcal{C} : u, v \in C\}$ to E and removing all edges of $E^- := \{(u, v) \in E \mid \exists C \in \mathcal{C} : u \in C \ \wedge \ v \notin C\}$ from E.

2 Refinement Techniques

Label Propagation. The label propagation algorithm is a local search heuristic that was initially proposed by Raghavan et al. [5] for community detection, as well as Karypis et al. [4] for hypergraph partitioning, and it was already successfully applied to the cluster editing problem [1]. The algorithm works in rounds. In each round, we visit the vertices in some

* 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.

order (random, increasing degree, or input) and move a vertex to an adjacent cluster that minimizes the number of required edit operations (ties are broken randomly). In addition to all adjacent clusters, we also consider isolating the vertex from its current cluster. The algorithm proceeds until we reach a predefined number of rounds or none of the vertices changed their cluster during a round. It can be initialized with an already existing solution or can be used to create an initial solution by initially assigning each vertex to its own cluster. For many problems, the algorithm converges within a few rounds. However, this was not the case here. We observed that for larger instances roughly 10% of vertices are moved during a round and most of them did not change the number of edits (these moves are called *zero-gain* moves). This naturally perturbs the solution and enables frequent improvements in later iterations of the algorithm (plateau search).

Clique Removal and Splitting. The next two refinement techniques were initially proposed by Bastos et al. [1] for the cluster editing problem. Both algorithms also work in rounds and in each round, we visit the clusters of the current solution in some order. The *clique removal* heuristic tries to remove a cluster from the solution by moving the vertices to an adjacent cluster that minimizes the number of edits (vertex isolation is also considered). The *clique splitting* technique chooses two non-adjacent vertices of the considered cluster as seed vertices and isolates them. Afterwards, we assign the remaining vertices to the cluster of one of the two seed vertices based on which induces less edit operations. If the applied moves increased our objective function, we revert them.

Vertex Swapping. The vertex swapping algorithm proceeds in a similar fashion as our other refinement algorithms (several rounds and we visit the vertices in some order). The algorithm moves a vertex speculatively into an adjacent cluster (could potentially increase the number of required edits) and then tries to move vertices out of the target cluster. We move a vertex out of the target cluster, if it decreases the number of required edits. If the overall operation improves the solution quality, we apply the moves. Otherwise, we revert them.

3 Variable Neighborhood Search

We have implemented two different algorithms that are based on the variable neighborhood search (VNS) heuristic [3]. The general idea of such algorithms is to start from an initial solution and then *perturb* the current solution (also called *shaking* in the literature) followed by *local search* phase to find a new local optimum. This process can be repeated until a predefined number of rounds or a time limit is reached.

Our first algorithm applies a perturbation operator to a large number of vertices (between 1% and 25% of the vertices) and then runs the refinement techniques described in Section 2. The second algorithm only perturbs a few vertices (between 1 and 5) and then executes the label propagation algorithm only initialized with the mutated vertices and their neighbors. We call the first one *global* and the second *highly-localized VNS*. The rationale behind this is that there are instances that converge very quickly and only specific mutations lead to additional improvements. For such instances, the global algorithm is a good choice, since bad decisions are reverted very quickly and the likelihood of mutating the right vertices is very high. On the other hand, there are instances that converge very slowly, and the perturbation of a large number of vertices can drastically degrade the solution quality such that our refinement techniques are not able to restore the initial solution quality. Thus,

performing highly-localized mutations and refinement becomes an important techniques to escape from local optima.

We run the VNS algorithms in an alternating fashion, in short bursts of 20 seconds each. However, we track the improvement made in the last burst, and prefer running the more promising algorithm next.

Initial Solution. Initially, each vertex is assigned to its own cluster. To compute an initial solution, we first run 100 rounds of our label propagation algorithm, then 20 rounds of our vertex swapping algorithm and last, 10 rounds of the clique removal and splitting technique. We repeat this several times (at most 100 times, but we abort after 10 seconds) and use the best solution found as input for our VNS algorithms.

Global Variable Neighborhood Search. Our global VNS algorithm performs one of the following two operations in each step: It either *intensifies* the current solution by applying the refinement techniques described in Section 2 or applies one of our three *perturbation* operators followed by an intensify operation. The decision which operation we choose next is based on the improvement history of the last 5 runs of the corresponding operation. The more an operator improves the solution quality compared to the other, the more likely it is that we execute it next.

We have implemented three different perturbation operators, which are also described in the work of Bastos et al. [1]. Each operator mutates between 1% and 25% of the vertices, which we choose uniformly and at random. The first operator, which we call *vertex isolator*, isolates a vertex. Our *vertex mover* operator, moves a vertex to a random adjacent cluster. The third operator, which we call *clique splitter*, isolates a vertex from its current cluster and then chooses one additional adjacent vertex from its previous cluster to form a new clique of size two.

In an intensify step, we perform 25 rounds of label propagation, 5 rounds of vertex swapping and 10 rounds of clique removal and splitting each. The vertex swapping and clique removal and splitting techniques are only executed with a probability of 50%. After a perturbation operation, we perform 50 rounds of label propagation, but abort early if the estimated improvement (based on the improvement history of label propagation) is greater than our initial edits before the perturbation operation plus 100. Further, we perform another 50 rounds of label propagation, if we are only 25 edits away from the initial number of edits. We perform the other refinement techniques similar to before.

Highly-Localized Variable Neighborhood Search. In each step, our highly-localized VNS algorithm selects between 1 and 5 vertices uniformly at random that are perturbed and, subsequently performs a highly-localized run of label propagation. For each of the selected vertices, we additionally choose an adjacent neighbor that we also perturb with a probability of 75%. As perturbation operator, we either isolate a vertex from its current cluster or move it to a random adjacent cluster. Afterwards, we perform 5 rounds of label propagation restricted to the perturbed vertices and their neighbors. If the overall operation improved the current solution, we apply all moves and otherwise, we revert them.

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 Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. KaPoCE - An Exact and Heuristic Solver for the Cluster Editing Problem. `https://github.com/kittobi1992/cluster_editing/tree/pace-2021`, 2021. `doi:10.5281/zenodo.4892524`.

3 Pierre Hansen and Nenad Mladenović. Variable Neighborhood Search. In *Handbook of Metaheuristics*, pages 145–184. Springer, 2003.

4 George Karypis and Vipin Kumar. Multilevel k-way Hypergraph Partitioning. Technical Report 98-036, University of Minnesota, 1998.

5 Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks. *Physical Review E*, 76(3):036106, 2007.