Abstract

We describe PACA-JAVA, an algorithm for solving the cluster editing problem submitted for the exact track of the Parameterized Algorithms and Computational Experiments challenge (PACE) in 2021. The algorithm solves the cluster editing problem by applying data-reduction rules, performing a layout heuristic, local search, iterative ILP verification, and branch-and-bound. We implemented the algorithm in the scope of a student project at the University of Bremen.

1 Introduction

An undirected and simple graph $G$ is a cluster graph if its components are cliques (complete graphs). In the cluster editing problem we are given an undirected graph $G$ and the problem is to find the minimum set of edge edits (additions or deletions) that turn $G$ into a cluster graph. The Parameterized Algorithms and Computational Experiments 2021-Challenge (PACE 2021) is devoted to the cluster editing problem [1]. As a student group at the University of Bremen (under the supervision of Mario Grobler, Roman Rabinovich and Sebastian Siebertz) we participated in the PACE challenge in the scope of a Bachelor project. We implemented an exact algorithm for the cluster editing problem in Java. In the following we describe our approach.

* 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.
2 Notation

All graphs in this work are undirected, unweighted and have no self-loops. We use standard graph notation. A leaf is a vertex of degree one. For a vertex $v$ we write $N(v)$ for the neighborhood of $v$ in $G$. A path on three vertices is denoted by $P_3$. We write $P_3(G)$ for the set of induced $P_3$s of $G$. If an edge shall not be removed by any following procedure or has been added, we call it permanent. Analogously, if it shall not be added or has been removed, we call it forbidden. We call two $P_3$s $uvw$ and $xyz$ almost disjoint if $|\{u,v,w\} \cap \{x,y,z\}| = 1$.

A clique is a subgraph whose vertices are pairwise adjacent. A cluster graph is a graph in which every connected component forms a clique. Equivalently, a cluster graph is a graph that contains no induced $P_3$s. In the cluster editing problem we are given an undirected graph $G$ and the task is to compute a set of edge modifications (addition or deletion) of minimum size that transforms $G$ into a cluster graph.

3 Basic Solver Layout

For an input graph $G$ we perform the following steps:

- We delete isolated vertices.
- We apply some data reduction rules to reduce the size of the instance. See Section 4.
- We solve each component individually and combine the solutions afterwards. To solve an individual component $C$, we first calculate $|P_3(C)|$. Based on graph statistics we either use an ILP or a branch-and-bound-algorithm (see Sections 5–7).

4 Data Reduction Rules

- If $N(v) \setminus \{w\} = N(w) \setminus \{v\}$ and $vw \in E(G)$ for $v,w \in V(G)$, then the edge $vw$ is made permanent. The vertices $u$ and $v$ are twins and are guaranteed to lie in one cluster of the solution by the result of [2].
- The Single-Path-Reduction-Rule starts by finding a leaf. Then the outgoing path $p$ gets followed until a vertex $v$ with degree$(v) > 2$ is reached. Afterwards every second edge on this path gets removed. This happens only if $|p| > 1$. It is easy to see that this is a valid data reduction rule.

5 Finding $P_3$s

We compute the set of $P_3$s by using the approach described in Spinner’ thesis [8]. For every $v \in V(G)$ and all $u,w \in N(u)$ with $u < w$ we check whether $uw \in E(G)$. The triple $uvw$ is a $P_3$ if and only if $uw \notin E(G)$. To speed up the calculation we used the matrix datastructure from the Jeigen wrapper [6]. To check if a triple is a $P_3$ instead of checking the graph, we can retrieve the information out of the matrix, which in the end accelerates the $P_3$ calculation significantly. If the ratio between the number of induced $P_3$s and edges is at most 0.005 or the graph has at most 8000 edges, we continue to process the instance via an ILP based approach. Otherwise, we go into a branch-and-bound approach.

6 ILP

We compute an ILP instance where we add for every pair $i,j \in V(G)$ in the graph a variable $x_{ij}$ to the ILP. The intended meaning of $x_{ij} = 0$ is that $i$ and $j$ belong to the same cluster (have distance 0). The objective is to minimize $\sum_{ij \in E(G)} x_{ij} + \sum_{ij \notin E(G)} (1 - x_{ij})$. 
Additionally, if an edge is permanent or forbidden, we restrict the variable accordingly (by setting it to 1 or 0, respectively). For every $P_3$ in our graph we add the constraint $x_{uv} + x_{vw} - x_{uw} \geq 0$. We solve the ILP with the open source ILP solver SCIP ([4], [5], [7]). We verify whether the returned solution is a valid solution for cluster editing (observe that our ILP only ensures that all initially present $P_3$s are edited). If this is not the case, we add a constraint for each newly generated $P_3$ and iterate this process until we found a valid solution. This iterative approach turned out to be much faster than adding constraints for all triples $u, v, w \in V(G)^3$.

7 Branching

In our branch-and-bound approach we first compute an upper $u$ and lower bound $\ell$ as follows.

7.1 Upper Bound

For the upper bound we start by calculating a two-dimensional layout based on the algorithm proposed by Fruchterman and Reingold [3]. We then calculate the clusters based on a distance parameter $\sigma$: we add an arbitrary vertex $v$ that is not yet part of a clique into a clique and then add all vertices $w$ with the property that no edge on the path from $v$ to $w$ is longer (in the layout) than $\sigma$ to the same clique. We repeat this until all vertices are grouped into a clique. We iterate over multiple possible distances. After each iteration we do post-processing. The post-processing is a local-search that consists of merging cliques and shifting one vertex into another clique if that decreases the cost of the solution. We do this exhaustively. For the shift vertex operation we also add one empty clique, to allow for the creation of a new clique.

7.2 Lower Bound

We calculate lower bounds in two different ways. If the graph is big or we have little time left, the lower bound is the number of disjoint and almost disjoint induced $P_3$s that we approximate with a greedy heuristic. Our second lower bound is the LP relaxation of our ILP. If by chance the LP solution is integral we verify if this is a valid solution for cluster editing and return it if this is the case.

7.3 Branching

After having calculated the upper bound $k_0 = u$ and lower bound $\ell$ (if $u = \ell$, then we already found a solution) we try to find a solution for $k_1 = u - 1, k_2 = u - 2, \ldots$ until no solution is found for some $k_i$. We then return the solution for $k_{i-1}$.

We branch based on the edges. For each vertex pair $uv$ contained in a $P_3$ we do the following: If $uv \in E(G)$, we set the status to forbidden, and try to solve the resulting graph recursively. If this does not succeed, we set the status to permanent and recurse again. If $uv \notin E(G)$ we proceed by first setting the edge to permanent and try solving the resulting graph recursively. If this does not succeed, we set the edge to forbidden and recurse again. We start with the edge contained in the most $P_3$. This leads to the needed edge modifications. If we change the status for an edge we use the so called transitive closure to resolve depending $P_3$s. Given $u, v, w \in V(G)$, if $vw$ and $uv$ are marked as permanent edges, we also mark $uw$ as a permanent edge, as there needs to be a edge or else we get a $P_3$. In each recursive step, we call the ILP if the graph is simple enough, otherwise we continue based on the procedure branch-and-bound as mentioned before. Furthermore, we use the lower bound to cut branches.
References


