Rest in PACE - Exact[*]

Valentin Bartier
G-SCOP, Grenoble INP, Univ. Grenoble-Alpes, Grenoble, France

Gabriel Bathie
École Normale Supérieure de Lyon, France

Nicolas Bousquet
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Marc Heinrich
University of Leeds, UK

Théo Pierron
LIRIS, CNRS, Université Claude Bernard Lyon 1, Université de Lyon, France

Ulysse Prieto
Independent Researcher

1 General description of the solver

Our solver is in a branch and bound (BB) algorithm. It first finds an initial upper bound on the optimal solution using an heuristic algorithm (for more details, see our heuristic solver description in the corresponding track). The algorithm then constructs solutions starting from the initial partial clustering where all the vertices are in a cluster of size 1. At each step, either it merges two clusters or definitively separate them. If the cost of a solution is above the upper bound, we cut the branch. Let us now give a few additional details.

First we start with a preprocessing. We

partition the graph into connected components that will be solved independently one after the other. We then apply some kernelization rules that reduce the size of the instances (for more details, see our kernelization algorithm description). In this first part, we will also label some pairs of vertices as "edges" if they necessarily belong to any optimal solution or "non edges" if they do not belong to an optimal solution (see Section 2 for more details). This labeling phase permits to avoid unnecessary branchings in the following phase of the algorithm.

Let us now describe the branching phase of the algorithm. At each step, we have a partition of the graph into partial clusters (the initial partition being that every vertex is alone in its cluster). At each step, we will select the two clusters maximizing the number of edges between them and branch to merge them or to separate them definitively. We then compare the "cost" of the current partial solution with the cost of the best known solution and cut the branch if the actual cost is at least the cost of the best solution. In Section 3, we detail our definition of cost of the actual solution which is a lower bound on the cost of any solution which can be obtained starting from our current partial solution.

2 Labeling (non) edges

Two adjacent vertices u, v are *i-twins* if the symmetric difference of their neighborhoods has size exactly i. Note that 0-twins are what is usually called in the literature true twins. One can easily prove that two 0-twins are always in the same cluster of an optimal solution.

[*] This work was supported by ANR project GrR (ANR-18-CE40-0032).

Two false twins are not necessarily in the same cluster of an optimal solution (consider e.g. a P_3). However, we can prove that if two false twins u, v are not in the same cluster, we can move any of them to the cluster of the other without increasing the cost of an optimal solution. So we can assume that all the false twins are in the same cluster and then we can label as edges pairs of false twins. Similarly, we can label as edge any pair of 1-twins.

One can wonder what can happen when i is increasing. If there is a triangle of 2-twins then we can prove that there exists an optimal solution where there are in the same cluster. Unfortunately, it is not enough to force all these edges. Indeed for the butterfly (two triangles sharing one vertex), there are two triangles of 2-twins but we cannot force all the edges of all the triangles of 2-twins to be in an optimal solution since the optimal solution contains two clusters. We can however prove that if there is a K_4 of 2-twins, these vertices are in the same cluster in any optimal solution.

Generalizing to any i becomes harder and harder since the size of the clique is increasing. Instead we use the following fact easier to prove: for every $i \leq 8$, if a vertex v has at least $4i - 1$ i-twins X then $v \cup X$ are in the same cluster in any possible optimal solution. Note that we do not make any assumption on the twinness of X or the existence of the edges in X. Our proof is computer assisted (the program runs in a few minutes for $i = 8$). We conjecture that this statements holds for any i. The function might not be tight. For instance, it only gives 7 for $i = 2$ but it might be true that 5 is enough.

3 Lower bounds

When can we ensure that the current branch will not be optimal and then cut the current branch? We can indeed cut it if the number of edited edges is at least the cost of the best known solution. We would like to cut bad branch "before" by ensuring that the cost will indeed go over the optimal solution without branchings.

To this end, we look at the current graph and see how many edges (at least) has to be edited to find a solution. One can note that, for each P_3, we have to edit at least one edge. So if we can find a collection of P_3 that pairwise intersect on at most one vertex, the cost of any solution will be at least the current cost plus the number of P_3. Unfortunately this rule is often too weak since we often need to edit more than $|E|/2$ edges in the instances and the obtained lower bound here is at most $|E|/2$. We can improve this rule by noting that, for every star $K_{1,\ell}$ the number of edits is at least $\ell - 1$.

Our algorithm computes greedily a collection of P_3 that it extends to stars. The collection is updated after each branching of the algorithm. If the number of already edited edges plus the cost of the star collection is at least the cost of the optimal solution, we cut the branch.

Linear Programming. For small instances (up to 120 vertices) we run a Linear Programming algorithm to compute a lower bound using fractional stars. We then round the solution to an integral solution whose size often matches the lower bound (which permits to conclude without the branching algorithm).

Forced pairs. There are a few cases where we can ensure that an edge will or will not be in the final solution, without computing twinness of vertices. Consider a partial solution containing two clusters C_1, C_2 such that the number of (non yet fixed) non-edges between C_1 and C_2 plus the cost of the actual solution is at least the cost of the best solution found so far. Then merging C_1 and C_2 cannot improve the current best solution, and thus we can delete all the C_1, C_2 edges and mark all the C_1, C_2 pairs as fixed non-edges. We have a total of 5 such rules that we apply after each two branchings.