Cluster Editing — Heuristic Solution*

Quyen Linh TA [1] orcid-ID.png

[1] *MIAGE — University Paris Dauphine*
PSL University — Apprenticship Track
75016 Paris, France

Abstract

Inspired and referenced by optimization algorithms for Cluster Editing in many different studies, the combination of local search and graph kernelization presented by Gabriel Bathie (ENS Lyon) is the method that I chose to contribute this project. The kernelization rules that I used in this project were proven by G.Bathie to be safe and effective by the results he achieved earlier. Based on his algorithm, I do implementation in Python.

Keywords: Kernelization — FPT Algorithms — Local Search Heuristic

1. LOCAL SEARCH

1.1. *Local Move*

In a specified time, group the input graph into clusters, with a natural number **n subdivided clusters**, all vertices belong to one cluster and the rest are empty clusters (decode by ***cluster size null*** in program).
I start with a random vertex alpha, a solution S, the equation returns a solution S' where vertex alpha is moved across a cluster C (is an empty cluster or nearest cluster) to reduce the cost of S'. It can therefore be implemented to run in $O(d(v)time)$, where $d(v)$ is the degree of v.

1.2. *Local Search*

Aiming to avoid these local minima by taking larger moves, by creating any subset X consisting of many vertices t, move X to empty clusters, then insert them back. Of course, to optimize this movement, I choose the *tvertices* that are close to each other to become a set X. Using the **BFSGreedy** algorithm allows me to search and execute the optimal movement (but I'm not sure it's the best yet).

Algorithm 1 Best First Search Greedy Heuristic

```
procedure BFSGREEDY(C, S, t)
    visited ← [false] * n
    for each u in C random order do
        if visited[u] then
            process to next node
        X ← selectBFS(C, u, visited, t)
        S' ← S
        for each v in X do
            S' ← isolation(v, S')
            for each v in X do
                S' ← greedyMove(v, S')
    return S'
```

* Released on February, 9th, 2022

Algorithm 2 Cluster Editing Solver

```
procedure SOLVER(G, t, w)
    G ← kernelize(G)
    for each connected component C_i do
        S_i ← trivial_solution(C_i)
    while not reached SIGTERM signal do                ▷ or reach iteration
        for each connected component C_i do
            j ← sample_weight(w_1, ..., w_l)
            S' ← bfs_greedy(C_i, S_i, t[j])
            if cost(S) < cost(S_i) then
                S_i ← S'
                w[j] ← w[j] + 1
    return S = ⋃_i S_i
```

2. KERNELIZATION RULES

1. Let u be a vertex with either a 1-neighbor or two adjacent 2-neighbors. If u has another neighbor v such that u and v have no common neighbor, then delete uv.

2. Let uvxw be an induced C4 where v and w have degree 2. Delete uv and wx.

3. Let u be a 3-vertex with neighbors a, b, c.

   - If ab, bc and ac are edges, a has degree 3 and b, c both have degree at most 5, delete all edges bx and cx for x not in (u, a, b, c).

   - If ac and bc are edges, ab is a non-edge, and a, b, c all have degree at most 3, delete all edges ax and bx for x not in (u, c).

4. Let K be a clique on k vertices such that each vertex outside of K has at most one neighbor in K. For all of u in K, denote by f(u) the number of neighbours of u outside of k. $K = u_i, ..., u_k \text{where} f(u_1) \leqslant ... \leqslant f(u_k)$. For every i in $[1, k]$, $\sum_{j=i+1}^{k} f(u_i) \leqslant \binom{k}{2} - \binom{i}{2}$ delete all edges with exactly one endpoint in K

2.1. *Visualization*