PACE Challenge 2021 Solver Description, from the Random Sampling group of the UNSW GraphAbility VIP project

Angus Ritossa
UNSW Sydney, Australia

Paula Tennent
UNSW Sydney, Australia

Tiana Tsang Ung
UNSW Sydney, Australia

Akshay Valluru
UNSW Sydney, Australia

1 Overview

The solver uses two different algorithms, which are described below. The first algorithm generally performs better on sparse instances, and the second on dense instances. The solver runs the first algorithm once, and the second algorithm until the 10 minutes runs out. It then outputs the best solution it found.

2 Sparse Algorithm

This algorithm relies on the fact that high-scoring solutions to sparse graphs will consist of many small connected components.

Components with at most 2 vertices

The optimal solution when we limit the component size to 2 vertices is precisely the maximum matching. Our solver uses the augmenting path component of the blossom algorithm [1] to find an approximate matching.

Components with at most 3 vertices

In this case, we use a randomised algorithm. The following algorithm is used to create many components of size 3.

Initially, all vertices are unmarked. Repeat the following while there are sets of three vertices we haven't processed yet:

1. Select a set of three unmarked vertices x, y, z such that there is an edge between all pairs of these.
2. Mark these three vertices and add them as a component in our final graph.

We then find a matching within all unmarked vertices to create components of size 2.

3 Dense algorithm

The dense algorithm uses a heuristic to build up a cluster graph. Let S be a set of components, initially empty. We process each vertex i from 1 to n, and either add it to an existing component in S, or create a new component containing just that vertex which we add to S. We decide which component to insert i into by calculating a *cost* for each component, and insert i to the component with the minimum cost.

We calculate the cost of adding i to component X as follows: Let e be the number of edges (i, u) in the input graph where $u < i$ (that is, u is already in a component). Let e' be the number of edges (i, u) where $u \in X$. Note that $e' \leq e$. The cost is defined as $(|X| - e') + (e - e')$. The first part of the formula is the number of edges we need to add if we insert i into component X, and the second part is the number of edges we need to remove (only considering edges to vertices we have already processed. Hence, this is a reasonable approximation of the cost of inserting i into component X.

We use this same method to decide the cost of creating a new component containing just i, and we insert i into the component with minimal cost. Once we have done this for all i, we have a cluster graph.

Naively this cluster graph can be constructed in $O(n^2)$ time, however it can be done in $O(n + m)$, where n is the number of vertices and m the number of edges, by noting that we only need to consider inserting a vertex into a component if it has at least one edge to that component.

We have one final step which is used to improve the cluster graph. For each vertex i, we see if it is better to move it to another component. This is done by removing it from its current component, and then computing the cost of adding it back to its current component or adding it to a different component. Again, we add it to the component with the minimum cost. This procedure is done 10 times for each vertex, as we found after 10 iterations very few improvements are found.

Now, the effectiveness of this algorithm changes by a significant amount depending on the order of the vertices in the first step. As such, we run this with different, random, orderings of the vertices and take the best cluster graph it produces.

References

1 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.