PACE 2018 : Team Resilience

N. S. Narayanaswamy
swamy@cse.iitm.ac.in

Rajesh Pandian M.
mrprajesh@cse.iitm.ac.in

R. Vijayaragunathan,
vijayr@cse.iitm.ac.in
Department of Computer Science and Engineering.
Indian Institute of Technology Madras, India.

1 May 2018

1 Introduction

In PACE 2018 we have made submissions for both Track A and Track C. The former is the exact track while the later is a heuristic track. The problem for this year PACE is Steiner Tree Problem.

Input	: Graph $G(V, E)$, Edge weights $W : E \rightarrow \mathbb{Z}^+$ and terminals $L \subseteq V$
Output	: Find the minimum weighted tree (Steiner tree) T of G that contains all terminals.

2 Exact Algorithm

The dynamic programming(DP) algorithm is from the book by Niedermeier [Nie06]. The recurrence relation of the DP is the following:

$$C(v, X) = \min_{\substack{u \in V \\ \emptyset \subset X_1 \subset X \setminus \{u\}}} C(u, X_1) + C(u, (X \setminus \{u\}) \setminus X_1) + distance(u, v)$$

The running time of this algorithm [1] is $O^*(3^k)$.

Data: Graph $G(V, E)$, edge weights $W : E \rightarrow \mathbb{Z}^+$ and terminals $L \subseteq V$
Result: Find the minimum Steiner tree $T(V', E')$ such that $L \subseteq V'$

$minCost$ = INT_MAX;
$minTree = \phi$;
FloydWarshall(G);
for $v \in L$ **do**
    $C(v, L) =$ ComputeTable(G, v, L);
    $cost = C(v, L).value$;
    $T = C(v, L).edges$;
    if $cost < minCost$ **then**
        $minCost = cost$;
        $minTree = T$;
    end
end
PrintValueAndTree($minTree$);

Algorithm 1: FPT Algorithm - Driver.

```
Data: Graph G(V,E), a vertex v and set X ⊆ L.
Result: Tree value and tree edges of T.
if |X_1| = 1 then
    Let u ∈ X_1;
    T.addEdges(Path(u,v));
    return d(u,v), T ;
end
minCost = INT_MAX;
while hasPartition(X_1, X) do
    X_2 = X \ X_1;
    for u ∈ V do
        X_11 = X_1 \ {u};
        X_22 = X_2 \ {u};
        C_X1(u, X_11) = ComputeTable(G, u, X_11);
        C_X2(u, X_22) = ComputeTable(G, u, X_22);
        value = C_X1(u, X_11) + C_X2(u, X_22) + d(u,v);
        if value < minCost then
            value = minCost;
            T.addEdges(C_X1.edges);
            T.addEdges(C_X2.edges);
            minTree = T;
        end
    end
end
return minCost, minTree;
```

Algorithm 2: ComputeTable

3 Heuristic Algorithm

Our heuristic algorithm has two algorithms executed one after another and outputs the best of the two solutions. The algorithm is developed based on the intuition that joining and the connecting the path of a closest terminals to exiting the Steiner tree could yield the optimal Steiner tree. We start of with an arbitrary terminal and repeat above process on all the remaining terminals. We execute this sub-routine on the all terminals and save the best Steiner tree. This is our first algorithm. We call it frog tongue or *sticky tongue* algorithm. Later, we found out that the solution was improving if we pick two terminals at a time instead of one. In order to find a potential Steiner vertex for the two terminal, we employ a clever strategy. It is called as intersecting closest vertices of the terminals. This is our second algorithm and it gave us better results on some more instances. It is described in detail in Algorithm [3] below.

3.1 Notation

- For any pair of vertices $u, v \in V$, $d(u,v)$ denotes the shortest distance between u and v and $path(u,v)$ denotes the sequence of edges in the shortest path.
- For any vertex $v \in V$ and set $S \subseteq V$, $d(v,S)$ denotes the shortest distance between v and the closest vertex $u \in S$ to v. Similarly, $path(v,S)$ denotes the sequence of edges in the shortest path.
- For any $F \subseteq E$, $V(F)$ denotes the set of vertices incident on the edges in F.

Data: Graph $G(V, E)$, edge weights $W : E \rightarrow \mathbb{Z}^+$ and terminals $L \subseteq V$
Result: Find an optimal Steiner tree $T(V', E')$ such that $L \subseteq V'$

Find a best tree T' using sticky tongue algorithm;
$minCost = W(T')$;
$minTree = T'$;
for $v \in V$ **do**
  $T_v =$ ConnectTwoTerminals(G, v, L);
  $cost = W(T_v)$;
  if $cost < minCost$ **then**
    $minCost = cost$;
    $minTree = T_v$;
  end
end
PrintValueAndTree($minTree$);

Algorithm 3: Heuristic Algorithm - Driver.

Data: Graph $G(V, E)$, a vertex v and terminal set L.
Result: Tree T.

$T = \phi$;
$S = \{v\}$;
$RT = L \setminus S$ `/* Remaining Terminals or unvisited terminals */`
while $RT \neq \phi$ **do**
  if $|RT| = 1$ **then**
    Find the shortest path from $v_1 \in RT$ to T;
    Add those edges to T;
    break;
  end
  Pick the two closest terminals v_1, v_2 to T;
  Compute $d_1 = d(v_1, S)$ and $d_2 = d(v_2, S)$;
  Let $cV_1 = \{u \in V \mid d(u, v_1) \leq d_1\}$;
  Let $cV_2 = \{u \in V \mid d(u, v_2) \leq d_2\}$;
  $cV = cV_1 \cap cV_2$;
  $value_1 = d_1 + d_2$;
  Let $u = \min_{x \in cV} d(x, v_1) + d(x, v_2) + d(x, S)$;
  $value_2 = d(u, v_1) + d(u, v_2) + d(u, S)$;
  if $cV \neq \emptyset$ *and* $value_2 < value_1$ **then**
    $S = S \cup V(path(u, S) \cup path(u, v_1) \cup path(u, v_2))$;
    $T = T \cup path(u, S) \cup path(u, v_1) \cup path(u, v_2)$;
  else
    $S = S \cup V(path(v_1, S) \cup path(v_2, S))$;
    $T = T \cup path(v_1, S) \cup path(v_2, S)$;
  end
  $RT = L \setminus S$;
end
return Tree T;

Algorithm 4: ConnectTwoTerminals

References

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 01 2006.