# Dynamic Topical Graph Summarization

Han Xiao,[1*]

[1]Department of Computer Science, University of Helsinki,

[*]To whom correspondence should be addressed; E-mail: han.xiao@cs.helsinki.fi.

**The document approaches the problem of summarizing topical interaction graph(e.g, email interaction network). We transform the original interaction graph into *interaction meta graph*, which is more interpretable and easier to model. Then we decompose the summarization problem into two steps, first optimizing locally and then globally. Experiment shows that ...**

## Motivation

Interaction happens all the time among real/virtual entities in the world. Email user exchanges information with his contacts by sending/receiving emails. Social network user interacts with his friends by sending/receiving messages, liking and commenting the feed posts. In online media, business news manifests the interaction among business entities(e.g, companies). For example, Apple and Samsung battle on patent infringement.

Given a series of interactions, it would be interesting to see what is happening in a bird's view. For example, a Facebook users might be curious to know what he has talked about, with whom and during the past 2 months. Or news readers want to know what are the major events about Google with its related companies for the last 10 years.

In this paper, we focus on summarizing text-based interaction networks. All the interactions in the network is characterized by text documents, for example email body in corporate email communications or instant messages in Facebook user communication networks.

## Meta Graph Framework

### Topical Interaction Graph

To represent interactions' textual content more compactly, we apply topic modeling techniques such as LDA (*1*) on all the text in the network and associate each interaction with a topic vector.

(Comment: I realized that other text representation models can be applied as well, such as Bag-of-Words model)

A *topical interaction graph* is defined as a directed graph $G_s = (V_s, E_s, \beta)$, where $V_s$ is a set of $N$ nodes and $E_s$ is a set of $M$ time-stamped pair-wise interactions among the nodes. Each interaction is associated with a topic distribution $\alpha$ of $L$ topics. $\beta$ is the global topic-to-token probability.

$E = \{(u_i, v_i, \alpha_i, t_i) | i = 1 \ldots M\}$

where $u_i, v_i \in V$, $\alpha_i \in \mathbb{R}^L$. $\beta \in \mathbb{R}^{L \times W}$

For example, in email interaction network, $V_s$ correspond to email addresses and $E_s$ correspond to the messages exchanged among $V_s$.

### Meta Interaction Graph

We transform dynamic topical graph, $G_s = (V_s, E_s)$ into *meta interaction graph* $G = (V, E)$ where $V = E_s$ and $E = \{(i, j) | i, j \in V \wedge i.t < j.t \wedge (i.u = j.u \vee i.v = j.u)\}$. The process is illustrated in Figure 1. The resulting graph is a DAG.

Using the email interaction example, $V$ corresponds to messages. Besides strong temporal order constraint, edges between $i, j \in V$ are created to model three patterns of information flow.

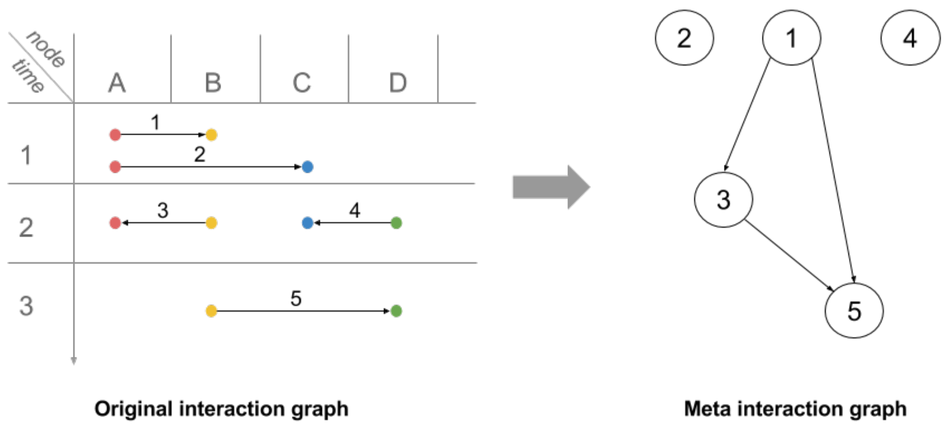

Figure 1: Converting topical interaction graph to meta interaction graph. In this example, there are 4 nodes and 5 interactions spanning 3 time slices. Note that interaction 2 and 4 are singleton. Edges (1, 5), (1, 3), (3, 5) captures the *relay, reply, distribute* patterns respectively.

1. *relay*: $i.u$ messaged $i.v$ and $i.v$ forwarded to $j.v$. In this case, $i.v = j.u$.
2. *reply*: $i.u$ messaged $i.v$ and $i.v$ replied to $i.u$. In this case, $i.v = j.u$ and $j.v = i.u$
3. *distribute*: $i.u$ messaged $i.v$ and later messaged $j.v$. In this case $i.u = j.u$ and $j.v \neq i.u$

Note that if two messages are connected, they are only potentially related.

## Practical issue

In practice, a message can be sent to more than two addresses at a time. In this case, we decompose the message into multiple disconnected interactions, each corresponds to one different receiver. For example, in Figure 1, *A* sent to a message to both *B* and *C* at time *1*. In the meta graph, two disconnected interactions *1* and *2* are created.

# Problem Formulation

The problem is to find $K$ events that best capture the interaction graph. In the meta graph framework, given $G = (V, E)$, an event is defined to be a subtree $T = (W, D)$, where $W \in V$

and $D \in E$. The root corresponds to event starting point. As time goes by, information flows and pushes event forward.

We divide the problem into two sub-problems. First, we calculate a list of candidate event trees, each rooted at a different node. Second, we select $K$ trees among the candidates that maximize the interaction coverage.

In the following two sections, we tackle the above sub problems respectively.

## Finding the Best Event

We first restrict us to find *the best* event rooted at given node. We assume a "good" event is characterized by 1), topical coherence, 2) relatively short period and 3) covering a large number of interactions. More formally, we define event to be optimal if it has the largest interaction coverage meanwhile under certain time and topic coherence constraints. Mathematically, $T = (W, D)$ is optimal if $|W|$ is maximized under the constraints $c_{time}(T) < U_1$ and $c_{topic}(T) < U_2$.

### Event Constraint

Time constraint can be modeled in two ways:

**Option 1**: Each edge in $E = (u, v)$ is weighted by the time difference between $u$ and $v$, $w(v, u) = v.t - u.t$. For an event $T = (W, D)$, we constrain the sum of its edge weights to be under some threshold. More formally, it's necessary for a good event $T$ to satisfy $\sum\limits_{(u,v) in D} w(v, u) \leq U$. However, such summation is not easy to interpret.

**Option 2**: We constrain all interactions of an event to be within a given time span. Time span of $T$ is defined $ts(T) = (min(\{v.t|v \in W\}), max(\{v.t|v \in W\}))$. A good event should satisfy $ts(T) \leq U$. Compared to Option 1, this one is easier to interpret. Thus, we stick to this option.

Topical coherence constraint can be modeled in two ways:

**Option 3**: topic divergence is modeled *locally*. Each edge is weighted by topical divergence between $u$ and $v$, $d(u.\alpha, v.\alpha)$. $d$ can be any vector distance measure, such as Kullback-Leibler divergence, Euclidean distance, etc. We define topic coherence of node sets $W$ as $td(W) = \frac{1}{|D|} \sum\limits_{(u,v) \in D} d(u.\alpha, v.\alpha)$. It's necessary for a good event $T$ to satisfy $td(T) \leq U$

**Option 4**: topic divergence is modeled *globally*, compared to Option 3. We define topic divergence of a set of nodes, $W$ as $td(W) = \frac{1}{|D|} \sum\limits_{u \in W} (d(u.\alpha, \bar{\alpha})$, where the *event topic vector* $\bar{\alpha} = \frac{1}{|W|} \sum\limits_{u \in W} u.\alpha$ and $d$ is some vector distance measure.

**Two Problems**

Option 2 can be easily treated by pruning the nodes that are outside of the time span relative to the root node. Thus, option 3 and 4 are remained, leading to two problem definitions.

**Problem 1**: given meta graph $G = (V, E, c)$, where $c : E \to \mathbb{R}$, root node, $r \in V$ and upper bound $U \in \mathbb{R}$, our goal is to find a subtree $T = (W, D)$ rooted at $r$ that maximizes:

$|W|$

s.t:

$td(W) = \sum\limits_{e \in D} c(e) \leq U$

We observe this problem is directly related to Prize Collecting Steiner Tree(PCST), more specifically the *Budget* version in (*2*).

Also, it can be transformed to *Net Worth Maximization* version in (*2*).

Using Lagrangian Relaxation, we rewrite the problem as:

find a subtree $T = (W, D)$ rooted at $r$ that maximizes

$|W| + \lambda(U - \sum\limits_{e \in D} c(e))$

where $\lambda$ is the Lagrange multiplier.

**Problem 2:** given a graph $G = (V, E)$, root node $r \in V$ and upper bound $U \in \mathbb{R}$, find a tree $T = (W, D)$ rooted at $r$, where $W \subseteq V, D \subseteq E$ that maximizes

$|W|$

s.t.

$td(W) = \sum_{u \in W} d(u.\alpha, \bar{\alpha}) \le U.$

where $\bar{\alpha} = \frac{1}{|W|} \sum_{u \in W} u.\alpha$ and $d$ be some vector distance measure.

### Finding $K$ Best Events

After we obtain the optimal subtrees rooted at each node in $G$, we aim to select $K$ subtrees among all the optimal subtrees that maximize the interaction coverage.

We define $T^*(G, r, U)$ the function that outputs the optimal subtree of $G$ rooted at $r$ under constrain parametrized by $U$. Therefore, we have the list of all optimal subtrees $\mathcal{T} = \{T^*(G, r, U) | r \in G.V\}$.

**Problem 3**: Given $G = (V, E)$, $\mathcal{T} = \{T^*(G, r, U) | r \in G.V\}$, $K \in \mathbb{Z}$, find $\mathcal{T}^{'} \subseteq \mathcal{T}$ that maximizes

$| \bigcup_{(W,D) \in \mathcal{T}^{'}} W|$

It is easy to observe that this problem is equivalent to maximum $K$-coverage problem.

## Algorithms

We describe three algorithms, one for each problem in this section.

### Problem 1

This problem can be seen as length-constrained maximum-sum problem (*3*). In (*3*), a pseudo dynamic programming algorithm is proposed for the special case where $G$ is a tree. We show that the algorithm can be adapted to DAG as well.

**Note**: the algorithm only guarantees optimality when edge weights are **positive**.

**Binary DAG**

Without loss of generality, we first solve the simpler case, binary DAG, where each node can have at most two children. General DAG can be converted to binary DAG, which will be explained later. We introduce node reward function $rw : V \to \{0, 1\}$. For nodes, $v$ in unconverted DAG, $w(v) = 1$. For newly-created nodes in the converted DAG, $v^{'}$, $w(v^{'}) = 0$.

Dynamic programming can be applied in the following way. Each node $u \in V$ has two hash tables $A_u$ and $D_u$. $A_u[i] = j$ denotes for the optimal tree rooted at $u$, $|W| = j$ and $\sum\limits_{e \in D} c(e) = i$. Initially, for each node $u \in V$, $A_u[0] = 1$. $D_u[i]$ is the set of nodes of the optimal tree corresponding to $A_u[i] = j$. Initially, for each node $u \in V$, $D_u[0] = \{u\}$.

If $u$ has only one child $v$ and $c(u, v) \leq U$, $A_u[c(u, v)] = 1, D_u[c(u, v)] = \{u, v\}$

If $u$ has two children $v, w$,

$$A_u[i] = max \begin{cases} max\{A_v[j] + A_w[i - j - c(u, v) - c(u, w)] + rw(i) | j \in A_v \wedge \\ (i - j - c(u, v) - c(u, w)) \in A_w \wedge (D_v[j] \cup D_w[i - j - c(u, v) - c(u, w)]) = \varnothing\} \textbf{ \#1} \\ A_v[i - c(u, v)] + rw(i) \text{ if } (i - c(u, v)) \in A_v \textbf{ \#2} \\ A_w[i - c(u, w)] + rw(i) \text{ if } (i - c(u, w)) \in A_w \textbf{ \#3} \end{cases} \tag{1}$$

$$A_u[k] = max \begin{cases} max\{A_v[i] + A_w[j] + r(u) | i + j + c(u, v) + c(u, w) = k\} \\ A_v[k - c(u, v)] + r(u) \\ A_w[k - c(u, w)] + r(u) \end{cases} \tag{2}$$

For different cases in the above equation, $D_u$ is updated accordingly:

$$D_u[i] = \begin{cases} D_v[j^*] \cup D_w[j^*] \cup \{u\} \text{ if } \textbf{\#1} \\ D_v[i - c(u, v)] \cup \{u\} \text{ if } \textbf{\#2} \\ D_w[i - c(u, w)] \cup \{u\} \text{ if } \textbf{\#3} \end{cases} \tag{3}$$

where $j^*$ is the optimal index for **#1**.

```
**Data**: $G, U, r$
**Result**: Maximum spanning tree rooted at $r$
A, D, BP = empty hash table;
**for** $u$ *in* $G.V$ **do**
    $A[u]$ = empty hash table with default value $-\infty$; $A[u][0]$ = 1;
    $D[u]$ = empty hash table; $D[u][0] = \{u\}$;
    $BP[u]$ = empty list;
**end**
**for** $u$ *in topological_sort*$(G.V)$ **do**
    **if** $u.children.length == 1$ **then**
        $v = u.children[0]$;
        **for** $j$ *in* $A[v]$ **do**
            $c = j + G.c(u, v)$;
            **if** $c \leq U \wedge A[u][c] < A[v][j] + rw(u)$ **then**
                $A[u][c] = A[v][j] + rw(u)$;$D[u][c] = D[v][j] \cup \{u\}$; $BP[u][c] = [(v, j)]$;
            **end**
        **end**
    **else**
        $v, w = u.children[0], u.children[1]; lc, rc = G.w(u, v), G.w(u, w)$;
        **for** $j$ *in* $A[v]$ **do**
            $c = j + lc$;
            **if** $c \leq U \wedge A[u][c] < A[v][j] + rw(u)$ **then**
                $A[u][c] = A[v][j] + rw(u)$;$D[u][c] = D[v][j] \cup \{u\}$; $BP[u][c] = [(v, j)]$;
            **end**
        **end**
        **for** $k$ *in* $A[w]$ **do**
            $c = j + rc$;
            **if** $c \leq U \wedge A[u][c] < A[w][k] + rw(u)$ **then**
                $A[u][c] = A[w][j] + rw(u)$;$D[u][c] = D[w][j] \cup \{u\}$; $BP[u][c] = [(w, k)]$;
            **end**
        **end**
        **for** $j$ *in* $A[v], k$ *in* $A[w]$ **do**
            $c = j + k + lc + rc$; $s = A[v][j] + A[w][k] + 2$;
            **if** $c \leq U \wedge A[u][c] < s \wedge A[v][j] \cap A[w][k] = \varnothing$ **then**
                $A[u][c] = s$; $D[u][c] = A[v][j] \cup A[w][k] \cup \{u\}$;
                $BP[u][c] = [(v, j), (w, k)]$;
            **end**
        **end**
    **end**
**end**
$c^* = argmax_i\{A[r][i] | i \in A[r]\}$; $s$ = empty stack; $t$ = empty tree;
**for** $u, c$ *in* $BP[r][c^*]$ **do** $s.append((r, v, c))$ ;
**while** $s.not\_empty$ **do**
    $u, v, c = s.pop()$;
    $t.add\_edge(u, v)$;
    **for** $w, c2$ *in* $BP[child][c]$ **do** s.push((v, w, c2)) ;
**end**
```

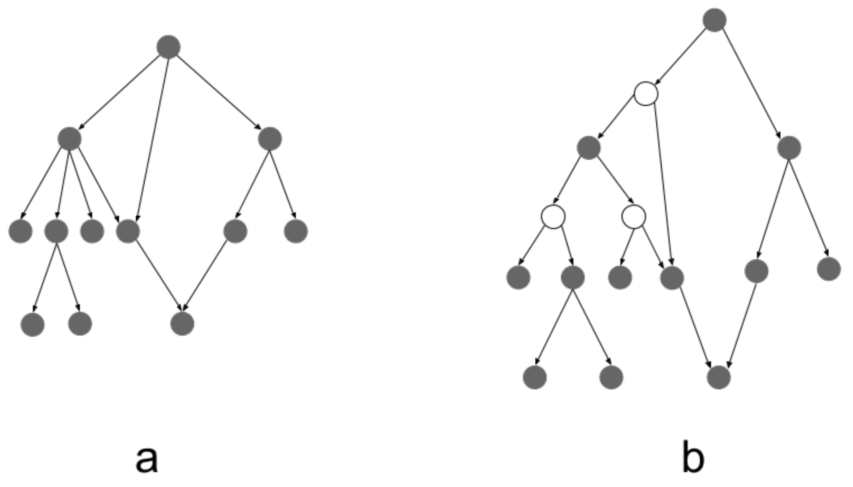

Figure 2: **a**: a general DAG, **b**: the final result after transforming multi-children nodes

**General DAG**

In order to run the above algorithm for general DAG, general DAG should be converted to binary DAG first.

The conversion is done by traversing the tree at top-down order and performing: for each node $u$, if it has more than 2 children: divide children nodes into pairs $p_1, \ldots, p_k$. For each pair $p_i$, create a new node $v$, set it as $p_i$'s parent and set $v$'s parent to be $u$. Set edge cost $c(u, v) = 0, c(u, p_{i,1}) = c(v, p_{i,1}), c(u, p_{i,2}) = c(v, p_{i,2})$ and node reward $rw(v) = 0$. Repeat the process for $u$ until $u$ has only 2 children. The process is illustrated in Figure 2.

To get the optimal answer for general DAG, perform the following procedure:

1. Convert the DAG into binary form.

2. Obtain the result of the binary DAG.

3. Convert the result back to the subtree of the original DAG by back tracing.

## Time Complexity

For the graph conversion procedure, it takes $O(|V|^2)$. For each node $u$ of degree $d(u)$, we create $O(d(u))$ nodes and edges. As $G = (V, E)$ is DAG, $\sum\limits_{u \in G.V} d(u) < |V|^2$.

For real-numbered edge cost, the dynamic programming procedure is pseudo polynomial. To reduce computation time in practice, we can apply fixed point arithmetic. As a special case, for integer edge cost, the running time is $O(|V|U^2)$ (*3*).

## Notes on PCST

This problem can be viewed as PCST as well. There exists many studies on approximating the PCST problem for general graph (*2, 4, 5*). However, we are dealing with DAGs. It would be interesting to ask how to solve PCST for DAGs.

**More to read.**

## Problem 2

Dynamic programming can be applied in this case as well because this problem shares similar structure as Problem 1. In fact, dynamic programming for Problem 2 only differs with Problem 1 in the way they calculate the cost for subtrees.

```
**Data**: $G, U, r$
**Result**: Maximum spanning tree rooted at $r$
A, D, BP = empty hash table;
**for** $u$ *in* $G.V$ **do**
    $A[u]$ = empty hash table with default value $-\infty$; $A[u][0]$ = 1;
    $D[u]$ = empty hash table; $D[u][0] = \{u\}$;
    $BP[u]$ = empty list;
**end**
**for** $u$ *in topological_sort*$(G.V)$ **do**
    **if** $u.children.length == 1$ **then**
        $v = u.children[0]$;
        **for** $j$ *in* $A[v]$ **do**
            $c = td(D[v][j] \cup \{u\})$;
            **if** $c \leq U \wedge A[u][c] < A[v][j] + rw(u)$ **then**
                $A[u][c] = A[v][j] + rw(u)$;$D[u][c] = D[v][j] \cup \{u\}$; $BP[u][c] = [(v, j)]$;
            **end**
        **end**
    **else**
        v, w = u.children[0], u.children[rw(u)]; **for** $j$ *in* $A[v]$ **do**
            $c = td(D[v][j] \cup \{u\})$;
            **if** $c \leq U \wedge A[u][c] < A[v][j] + rw(u)$ **then**
                $A[u][c] = A[v][j] + rw(u)$;$D[u][c] = D[v][j] \cup \{u\}$; $BP[u][c] = [(v, j)]$;
            **end**
        **end**
        **for** $k$ *in* $A[w]$ **do**
            $c = td(D[w][k] \cup \{u\})$;
            **if** $c \leq U \wedge A[u][c] < A[w][k] + rw(u)$ **then**
                $A[u][c] = A[w][j] + rw(u)$;$D[u][c] = D[w][j] \cup \{u\}$; $BP[u][c] = [(w, k)]$;
            **end**
        **end**
        **for** $j$ *in* $A[v], k$ *in* $A[w]$ **do**
            $c = td(D[v][j] \cup D[w][k] \cup \{u\})$;
            $s = A[v][j] + A[w][k] + 2$;
            **if** $c \leq U \wedge A[u][c] < s \wedge A[v][j] \cap A[w][k] = \varnothing$ **then**
                $A[u][c] = s$; $D[u][c] = A[v][j] \cup A[w][k] \cup \{u\}$;
                $BP[u][c] = [(v, j), (w, k)]$;
            **end**
        **end**
    **end**
**end**
$c* = argmax_i\{A[r][i] | i \in A[r]\}$; $s$ = empty stack; $t$ = empty tree;
**for** $u, c$ *in* $BP[r][c^*]$ **do** $s.append((r, v, c))$ ;
**while** $s.not\_empty$ **do**
    $u, v, c = s.pop()$;
    $t.add\_edge(u, v)$;
    **for** $w, c2$ *in* $BP[child][c]$ **do** s.push((v, w, c2)) ;
**end**
```

## Problem 3

Problem 3 is equivalent to maximum $K$-coverage problem. We can apply the greedy algorithm in (*6*), which shows greedily constructed solutions are guaranteed to be within a factor of $1 - \frac{1}{e}$ of the optimal solution.

## Baseline

### Random growing tree

The candididate event tree is grown from root $r$ by randomly selecting edges. At each iteration, an edge is randomly drawn from the cut set defined on the current set of nodes in the tree. The process stops until the cost of tree exceeds some threshold. One undo step is required to ensure the tree cost stays under the threshold. The pseudo code is described in Algorithm 3.

```
Data: $G, U, r$
Result: Randomly grown tree rooted at $r$
$T = (W, D), W = \{r\}, D = \{\}$;
while $cost(T) \leq U$ do
    $(u, v) = random\_sample(cutset(W, G))$;
    $W = W|\{v\}$;
    $D = D|\{(u, v)\}$;
end
$W = W - \{v\}$;
$D = D - \{(u, v)\}$;
return $T$;
```

**Algorithm 3:** Random growing tree algorithm for DAG

### Greedy growing tree

Greedy growing tree algorithm is the same as random growing tree algorithm, except for how it selects edges. At each iteration, the edge with the least cost is selected. The process is described in Algorithm 4.

```
**Data**: $G, U, r$
**Result**: Greedily grown tree rooted at $r$
$T = (W, D), W = \{r\}, D = \{\}$;
**while** $cost(T) \leq U$ **do**
    $(u, v) = argmax_{(u,v) \in cutset(W,G)}\{c(u, v)\}$;
    $W = W|\{v\}$;
    $D = D|\{(u, v)\}$;
**end**
$W = W - \{v\}$;
$D = D - \{(u, v)\}$;
**return** $T$;
```

**Algorithm 4:** Greedy growing tree algorithm for DAG

## Other Application

Besides communication datasets(such as Enron), it can be applied to other types of datasets. For example, we observe that news articles usually contain a set of mentioned entities. Two articles mentioning a common set of entities are likely to be relevant, thus a edge should be created to connect them in the meta interaction graph.

## Questions

1. **Dropping time constraint**: Is it possible to drop time constraint? If topic drift exists, then problem 1 and problem 2 implicitly encourages shorter period event.

## PCST-DAG Hardness Proof

Here we present a polynomial reduction from Minimum Set Cover problem to Prize Collecting Steiner DAG.

Minimum Set Cover (SC)

INSTANCE: A collection $C$ of subsets of a finite set $S$ and an integer $k$.

QUESTION: Is there a set cover for $S$, i.e. a subset $C^{'} \subseteq C$ such that every element in $S$ belongs to at least one member of $C^{'}$ and the cardinality of $C^{'}$ is not greater than $k$?

Prize Collecting Steiner DAG (PCSD)

INSTANCE: A graph $G(V, E)$, a cost function $c : E \to \mathcal{R}$ and a reward function $r : V \to \mathcal{R}$ and $U \in \mathcal{R}$ and $r \in V$.

SOLUTION:: A subtree $T$ of $G$ whose cost sum is bounded above by $U$.

Consider $SC(S, C, m)$, where $S = \{s_i | i = 1 \cdots n\}, C = \{c_i | i = 1 \cdots m\}$. Construct a directed graph $G = (V, E)$ where $V = \{u, c_1, \cdots c_m, s_1, \cdots s_n\}$ and $E$ consists of

- $(u, c_i) \in E$ for all $i$
- $(c_i, s_i) \in E$ if $s_i \in c_i$

Cost function $c(e) = 1$. Reward function is defined as:

- $r(u) = N$
- $r(c_i) = 1$ for all $i$
- $r(s_i) = N$ for all $i$

where $N > k$

**Lemma 1**: If $SC(S, C, k)$ has a solution, $PCSD(G, u, m + n) > (n + 1)N$

Without loss of generality, let $C_i, \cdots, C_k$ be the set cover for $S$. We can construct a tree $T(V^{'}, E^{'})$ such that reward sum $r(T) > (n+1)N$ and cost sum $c(T) \leq m+n$ in the following way:

- $V^{'} = u, c_i \cdots c_k, s_i \cdots s_n$
- $(u, c_i) \in E^{'}$ for $i = 1 \cdots k$

- $(c_1, s_j) \in E^{'}$ if $s_j \in C_1$
- $(c_i, s_j) \in E^{'}$ if $s_j \in C_i \setminus \bigcup_{p=1 \cdots j-1} C_p$

**Lemma 2**: If $PCSD(G, u, m+n) > (n+1)N$, $SC(S, C, k)$ has a solution.

Denote the optimal tree of $PCSD(G, u, m+n)$ as $T(V^{'}, E^{'})$. $r(T) > (n+1)N$ implies $s_j \in V^{'}$ for all $j$. $T$ is a tree implies for each $s_j$, $(c_i, s_j) \in E^{'}$ for some $i$. Therefore the subsets represented by $c_i \in V^{'}$ covers the set.

Thus, PCSD is NP-hard.

## References and Notes

1. D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation (2003).
2. D. S. Johnson, M. Minkoff, S. Phillips, The prize collecting steiner tree problem: theory and practice (2000).
3. H. C. Lau, T. H. Ngo, B. N. Nguyen, Finding a length-constrained maximum-sum or maximum-density subtree and its application to logistics (2006).
4. M. X. Goemans, D. P. Williamson, A general approximation technique for constrained forest problems (1995).
5. M. X. Goemans, D. P. Williamson, The primal-dual method for approximation algorithms and its application to network design problems (1997).
6. D. S. Hochbaum, A. Pathria, Analysis of the greedy approach in problems of maximum k-coverage (1998).

## 1 Primal-dual forumation for PCST-DiGraph

Integer program(IP) of PCST on directed graph.

$$\text{min:} \sum_{e \in E} c_e x_e + \sum_{T \subset V : r \notin T} z_T (\sum_{i \in T} \pi_i)$$

subject to:

$$x(\delta^-(S)) + \sum_{T \supseteq S} z_T \geq 1 \qquad S \in V; r \notin S$$

$$\sum_{T \subset V; r \notin T} z_T \leq 1$$

$$x_e \in \{0,1\}$$

$$z_T \in \{0,1\} \qquad T \subset V; r \notin T$$

1. $x_e = 1$ if $e$ is chosen and $0$ otherwise
2. $z_T$: set of unspanned nodes. $z_T = 1$ if forall $v \in T$ is not spanned and forall $v \notin T$ is spanned
3. $\delta^-(S) = \{(i,j) \in E | j \in S, i \notin S\}$
4. $x(F) = \sum_{e \in F} x_e$: number of edges in $F$ that are selected

**Why this IP works for the directed tree case?**

Suppose there is a solution that satisfies the constraints but is actually infeasible.

```
- DAG: try counter-cases
  - S = (B, A) and B -> A and C -> A: violates
  - disconnected tree? violates
  - diamond? same argument as above
- general DiGraph:
  - circle? similar to diamond(remove one edge)
```

Dual of the linear relaxation of IP

$$\text{max: } \sum_{S: r \notin S} y_S$$

subject to:

$$\begin{aligned}
&\sum_{S: e \in \delta^-(S)} y_S \leq c_e && e \in E \\
&\sum_{S \subseteq V} y_S \leq \sum_{i \in T} \pi_i && T \subset V; r \notin T \\
&y_S \geq 0 && S \subset V; r \notin S
\end{aligned}$$

## 2 Greedy algorithm for PCST-DiGraph

```
**Data**: $G = (V, E)$, root $r$, edge cost $c_{ij}$, vertex penalties $\pi_i$
**Result**: A tree $F^{'}$ rooted at $r$, uncovered vertices $X$
$F \leftarrow \emptyset$;
$\mathcal{C} \leftarrow \{\{v\} : v \in V\}$;
for $v \in V$ do
    $d(v) \leftarrow 0$ ;
    $w(\{v\}) \leftarrow 0$ ;
    if $v$ = $r$ then
        $\lambda(\{v\}) = 0$;
    else
        $\lambda(\{v\}) = 1$;
    end
end
while $\exists C \in \mathcal{C} : \lambda(C) = 1$ do
    Find edge $e = (i, j)$ with $i \in C_p \in \mathcal{C}, j = C_q.root, C_q \in \mathcal{C}, C_p \neq C_q$ that minimizes
    $\epsilon_1 = \frac{c_e - d(j)}{\lambda(C_q)}$;
    Find $\hat{C} \in \mathcal{C}$ with $\lambda(\hat{C}) = 1$ that minimizes $\epsilon_2 = \sum_{i \in \hat{C}} \pi_i - w(\hat{C})$;
    $\epsilon = min(\epsilon_1, \epsilon_2)$;
    for $C \in \mathcal{C}$ do
        $d(C.root) \leftarrow d(C.root) + \epsilon \dot{\lambda}(C)$ ;
        $w(C) \leftarrow w(C) + \epsilon \dot{\lambda}(C)$ ;
    end
    Comment: implicitly set $y_C \leftarrow y_C + \epsilon\lambda(C)$ for all $C \in \mathcal{C}$ ;
    if $\epsilon = \epsilon_2$ then
        $\lambda(\hat{C}) \leftarrow 0$;
        Mark all unlabeled vertices in $\hat{C}$ with label $\hat{C}$;
    else
        $F \leftarrow F \cup \{e\}$;
        $C \leftarrow C \cup \{C_p \cup C_q\} - \{C_p\} - \{C_q\}$;
        $w(C_p \cup C_q) \leftarrow w(C_p) + w(C_q)$;
        if $r \in C_p \cup C_q$ then
            $\lambda(C_p \cup C_q) \leftarrow 0$;
        else
            $\lambda(C_p \cup C_q) \leftarrow 1$;
        end
    end
end
Derive $F^{'}$ by removing as many edges in $F$ as possible while the following properties
hold: (1), every unlabeled vertex is connected to $r$; (2) if vertex $v$ with label $C$ is
connected to $r$, then so is every vertex with label $C^{'} \supseteq C$;
return $F^{'}$ and uncovered vertices $X$ ;
```

**Algorithm 5:** Greedy algorithm for prize collecting steiner tree on directed graph

## 3 PCST-DAG proof

Justification on the modification:

1. $d(i) = \sum_{\{S \in V | \text{can form a tree} \&\& S.r=i\}} y_S$: by induction, at kth iteration, if it's not root, the value does not change at (k+1)th iteration. If it's root, at k+1 iteration, one of the $y_S$ increases and $d(i)$ increases the same amount.

2. Optinally, $d(i) = \sum_{\{S \in V | \text{S forms a tree at sometime} \&\& S.r=i\}} y_S$

3. $\sum_{e \in \delta^-(S)} y_S = d(j)$, where $e = (i, j)$: suppose if there is a tree formed by $S^{'}$, $T(S^{'})$ such that $y_{S'} > 0$ and $j$ is not root in $T(S^{'})$, then the algorithm has formed such a tree before, which is a contradiction because $j$ is a root in the current tree. Because it's impossible for a node to become a root if it has become a non-root.

Approximation guarantee proof:

Since $\sum_{S \in V} y_S \leq Z^*_{LP} \leq Z^*_{IP}$, we want to know the relationship between $\sum_{e \in F'} c_e + \sum_{i \in X} \pi_i$ and $\sum_{S \in V} y_S$.

1. left-hand side = $\sum_S y_S |F^{'} \cap \delta^-(S)| + \sum_j \sum_{S \subseteq C_j} y_S$

2. At each iteraction, left-hand side increases by $\epsilon(\sum_{v \in N_a} din_v + |N_d|) = \epsilon(\sum_{v \in N_a - N_d} din_v + |N_d|) = \epsilon(|N_a - N_d| + |N_d|) = \epsilon|N_a|$

3. At each iteraction, right hand side increases $\epsilon|N_a|$

4. So it's optimal

**It cannot be true for general directed graph because it's NP-hard to solve optimally. Then there must be some problem either in the formulation or the proof**

## 4 Iterative algorithm for Budget-DiGraph

```
Data: $G = (V, E)$, root $r$, edge cost $c_{ij}$, budget $B$
Result: A tree $F^{'}$ rooted at $r$
$\lambda_1 \leftarrow 0$;
$\lambda_2 \leftarrow \sum_{e \in E} c_e$;
while $|\lambda_2 - \lambda_1| > \epsilon$ do
    $\lambda = \frac{\lambda_1+\lambda_2}{2}$ ;
    $\pi_i = \lambda$ for all $i \in V$;
    $T^{'} = PCST\_DiGraph(G, r, c_{ij}, \pi_i)$ ;
    if $T^{'}.cost \neq B$ then
        if $T^{'}.cost > B$ then
            $\lambda_2 = \lambda$ ;
        else
            $\lambda_1 = \lambda$ ;
        end
    else
        break ;
    end
end
return $T^{'}$ ;
```

Problem found:

1. for general DiGraph: an example where the greedy solution yields value(the lower bound) **greater** than optimal primal solution. If LP is right, then the way I calculate the LB is wrong(the algorithm can be wrong).

## 5 Primal-dual forumation for PCST-DAG

Integer program(IP) of PCST on DAG.

$$\text{min: } \sum_{e \in E} c_e x_e + \sum_{T \subset V: r \notin T} z_T \left(\sum_{i \in T} \pi_i\right)$$

subject to:

$$x(\delta^-(\{S.root\})) + \sum_{T \supseteq S} z_T = 1 \quad S \text{ can form a tree}; r \notin S$$

$$\sum_{T \subset V; r \notin T} z_T \leq 1$$

$$x_e \in \{0,1\}$$

$$z_T \in \{0,1\} \qquad T \subset V; r \notin T$$

Dual of the linear relaxation of IP

$$\text{max: } \sum_{S: r \notin S} y_S$$

subject to:

$$\sum_{S: e \in \delta^-(S)} y_S \leq c_e \qquad e \in E$$

$$\sum_{S \subseteq V} y_S \leq \sum_{i \in T} \pi_i \quad T \subset V; r \notin T$$

$$y_S \geq 0 \qquad S \subset V; r \notin S$$