Online Social Networks and Media

Team Formation in Social Networks
Thanks to Evimari Terzi

ALGORITHMS FOR TEAM FORMATION
Team-formation problems

- Given a **task** and a set of **experts** (organized in a **network**) find the subset of experts that can **effectively** perform the task

- **Task**: set of required skills and potentially a budget

- **Expert**: has a set of skills and potentially a price

- **Network**: represents strength of relationships
Insider

Security expert

Organizer

Electronics expert

Co-organizer

Mechanic

Mechanic

Pick-pocket thief

Explosives expert

Con-man

Acrobat
Applications

- Collaboration networks (e.g., scientists, actors)
- Organizational structure of companies
- LinkedIn, UpWork, FreeLance
- Geographical (map) of experts
Simple Team formation Problem

- **Input:**
  - A task $T$, consisting of a set of skills
  - A set of candidate experts each having a subset of skills

$$T = \{\text{algorithms, java, graphics, python}\}$$

<table>
<thead>
<tr>
<th></th>
<th>Alice</th>
<th>Bob</th>
<th>Cynthia</th>
<th>David</th>
<th>Eleanor</th>
</tr>
</thead>
<tbody>
<tr>
<td>skills</td>
<td>{algorithms}</td>
<td>{python}</td>
<td>{graphics, java}</td>
<td>{graphics}</td>
<td>{graphics, java, python}</td>
</tr>
</tbody>
</table>

- **Problem:** Given a task and a set of experts, find the smallest subset (team) of experts that together have all the required skills for the task
Set Cover

• The Set Cover problem:
  – We have a universe of elements $U = \{x_1, ..., x_N\}$
  – We have a collection of subsets of $U$, $S = \{S_1, ..., S_n\}$, such that $\bigcup_i S_i = U$
  – We want to find the smallest sub-collection $C \subseteq S$ of $S$, such that $\bigcup_{S_i \in C} S_i = U$
  - The sets in $C$ cover the elements of $U$
The Simple Team Formation Problem is a just an instance of the **Set Cover** problem

- **Universe** $U$ of elements = Set of all skills
- **Collection** $S$ of subsets = The set of experts and the subset of skills they possess.

\[
T = \{ \text{algorithms, java, graphics, python} \}
\]
Complexity

• The Set Cover problem are NP-complete
  – What does this mean?
  – Why do we care?

• There is no algorithm that can guarantee finding the best solution in polynomial time
  – Can we find an algorithm that can guarantee to find a solution that is close to the optimal?
  – Approximation Algorithms.
A simple approximation ratio for set cover

- Any algorithm for set cover has approximation ratio
  \[ \alpha = |S_{\text{max}}| \]
  where \( S_{\text{max}} \) is the set in \( S \) with the largest cardinality

- Proof:
  - \( \text{OPT}(X) \geq N/|S_{\text{max}}| \Rightarrow N \leq |S_{\text{max}}| \text{OPT}(X) \)
  - \( \text{ALG}(X) \leq N \leq |S_{\text{max}}| \text{OPT}(X) \)

- This is true for any algorithm.
- Not a good bound since it may be that \( |S_{\text{max}}| = O(N) \)
An algorithm for Set Cover

• What is the most natural algorithm for Set Cover?

• **Greedy**: each time add to the collection $C$ the set $S_i$ from $S$ that covers the most of the remaining uncovered elements.
The GREEDY algorithm

**GREEDY(U,S)**

\[ \begin{align*}
X &= U \\
C &= \{\} \\
\text{while } X \text{ is not empty do} \\
&\quad \text{For all } S_i \in S \text{ let } \text{gain}(S_i) = |S_i \cap X| \\
&\quad \text{Let } S_* \text{ be such that } \text{gain}(S_*) \text{ is maximum} \\
&\quad C = C U \{S_*\} \\
&\quad X = X \setminus S_* \\
&\quad S = S \setminus S_*
\end{align*} \]

The number of elements covered by \( S_i \) not already covered by \( C \).
Greedy is not always optimal

Alice
C, C++, Unix

Bob
C++, Unix, Java

Charlie
C, C++, Java, Python

Eleanor
Python, Joomla

Required Skills
C, C++, Unix, php, Java, Python, Joomla

David
php, Java, Python
Greedy is not always optimal

A different representation
Greedy is not always optimal

Optimal
Size 3 Set Cover
Greedy is not always optimal
Greedy is not always optimal

Optimal

Alice → C → C++ → Unix → php → Java → Python → Joomla

Greedy

Alice → C → C++ → Unix → php → Java → Python → Joomla
Greedy is not always optimal
Greedy is not always optimal
Greedy is not always optimal
Greedy is not always optimal

- Selecting Charlie is useless since we still need Alice and David.
- Alice and David cover together a superset of the skills covered by Charlie.
Good news: GREEDY has approximation ratio:

\[ \alpha = H(|S_{\text{max}}|) = 1 + \ln|S_{\text{max}}|, \quad H(n) = \sum_{k=1}^{n} \frac{1}{k} \]

\[ \text{GREEDY}(X) \leq (1 + \ln|S_{\text{max}}|)\text{OPT}(X), \text{ for all } X \]

The approximation ratio is **tight** up to a constant

– Tight means that we can find a counter example with this ratio

\[ \text{OPT}(X) = 2 \]
\[ \text{GREEDY}(X) = \log N \]
\[ \alpha = \frac{1}{2}\log N \]
Team formation in the presence of a social network

- Given a **task** and a set of **experts** organized in a **network** find the subset of experts that can **effectively** perform the task.

- **Task**: set of required skills

- **Expert**: has a set of skills

- **Network**: relationships and their strength

- **Effectively**: There is **good communication** between the team members
  - What does **good** mean? E.g., all team members are connected.
Coverage is NOT enough

T={\textit{algorithms, java, graphics, python}}

A, B, C form an effective group that can communicate

A, E can no longer perform the task since they cannot communicate

Communication: the members of the team must be able to efficiently communicate and work together
How to measure effective communication?

- **Diameter** of the subgraph defined by the group members

The longest shortest path between any two nodes in the subgraph

\[ \text{diameter} = 1 \]
How to measure effective communication?

- **MST (Minimum spanning tree)** of the subgraph defined by the group members

  The total weight of the edges of a tree that spans all the team nodes

  \[ \text{MST} = 2 \]
Problem definition \textbf{(MinDiameter)}

- Given a \textit{task} and a \textit{social network} $G$ of experts, find the subset \textit{(team)} of experts that can \textit{perform} the given task and they define a subgraph $G'$ in $G$ with the \textit{minimum diameter}.

- Problem is \textit{NP-hard}

- Equivalent to the \textbf{Multiple Choice Cover} (MCC)
  - We have a set cover instance \((U, S)\), but we also have a \textit{distance matrix} $D$ with distances between the different sets in $S$.
  - We want a cover that has the \textit{minimum diameter} (minimizes the largest pairwise distance in the cover)
The **RarestFirst** algorithm

- Compute all shortest path distances in the input graph \( G \) and create a new **complete** graph \( G_C \)
- Find **Rarest** skill \( \alpha_{\text{rare}} \) required for a task
- \( S_{\text{rare}} \) = group of people that have \( \alpha_{\text{rare}} \)
- Evaluate star graphs in \( G_C \), centered at individuals from \( S_{\text{rare}} \)
- Report cheapest star

**Running time:** **Quadratic** to the number of nodes

**Approximation factor:** \( 2 \times \text{OPT} \)
The RarestFirst algorithm

\[ T = \{ \text{algorithms, java, graphics, python} \} \]

\[ \{ \text{graphics, python, java} \} \]

\[ \{ \text{algorithms, graphics} \} \]

Skills:
- algorithms
- graphics
- java
- python

\[ \alpha_{\text{rare}} = \text{algorithms} \]

\[ S_{\text{rare}} = \{ \text{Bob, Eleanor} \} \]
The RarestFirst algorithm

$T = \{ \text{algorithms, java, graphics, python} \}$

$\alpha_{\text{rare}} = \text{algorithms}$

$S_{\text{rare}} = \{ \text{Bob, Eleanor} \}$

Diameter = 1

Skills:

- algorithms
- graphics
- java
- python
Analysis of **RarestFirst**

- The diameter is
  - either $D = d_k$, for some node $k$,
  - or $D = d_{\ell k}$ for some pair of nodes $\ell, k$

- Fact: $\text{OPT} \geq d_k$
- Fact: $\text{OPT} \geq d_\ell$

$$D \leq d_{\ell k} \leq d_\ell + d_k \leq 2\times \text{OPT}$$
Problem definition (MinMST)

- Given a task and a social network $G$ of experts, find the subset (team) of experts that can perform the given task and they define a subgraph $G'$ in $G$ with the minimum MST cost.

- Problem is NP-hard
- Follows from a connection with Group Steiner Tree problem
The Steiner Tree problem

- Graph $G(V,E)$
- Partition of $V$ into $V = \{R,N\}$
- Find $G'$ subgraph of $G$ such that $G'$ contains all the required vertices $(R)$ and $\text{MST}(G')$ is minimized
  - Find the cheapest tree that contains all the required nodes.
The Enhanced Steiner algorithm

Put a large weight on the new edges (more than the sum of all edges) to ensure that you only pick one for each skill.

\[ T = \{ \text{algorithms, java, graphics, python} \} \]

MST Cost = 1

Add the skills as new nodes in the graph, connected to the graph nodes that have the skill.

Solve the Steiner Tree on this graph, with the skill nodes being required.
The **CoverSteiner** algorithm

\[ T = \{ \text{algorithms, java, graphics, python} \} \]

1. Solve **SetCover**
2. Solve **Steiner**

**MST Cost = 1**
How good is **CoverSteiner**?

1. Solve **SetCover**
2. Solve **Steiner**

T = \{algorithms, java, graphics, python\}

\{graphics, python, java\} \quad \{algorithms, graphics\}

\{python, java\} \quad \{python\}

MST Cost = Infty
References

Theodoros Lappas, Kun Liu, Evimaria Terzi, Finding a team of experts in social networks. KDD 2009: 467-476