2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)

Template-Driven Team Formation

Spiros Apostolou, Panayiotis Tsaparas
Department of Computer Science & Engineering
University of Ioannina
Ioannina, Greece
sapostol,tsap@cs.uoi.gr

Evimaria Terzi
Department of Computer Science
Boston University
Boston, USA
evimaria@bu.edu

***Abstract*—The *team-formation* problem on social networks asks for a team of individuals that collectively possess the skills to perform a task and have low communication cost, as measured by their distances in the social network. Most related work assumes a flat structure in the team, where team members are all indistinguishable. However, in practice, teams often have complex structures and deep hierarchies, and members with distinct roles in these structures. In this paper, we consider the *Template-Driven Team Formation* problem, where given a fixed template structure for the team, in the form of a graph, and a designated role for each node in the template, we ask for workers that can fill the roles in the template, while minimizing the communication cost along the template edges. Although the problem is in general NP-hard, there are variants of the problem that can be solved optimally using dynamic programming. For the general case, we provide approximation and heuristic polynomial-time algorithms. We experiment on real data and we demonstrate that our heuristic algorithms perform well in practice while being significantly more efficient. Our case studies highlight the quality of the teams produced by our algorithms.**

***Index Terms*—team formation, template graph, algorithms**

I. Introduction

Over the last few years, the *team-formation* problem has received an increasing interest from researchers and practitioners[1] alike. The popularity of online labor markets (e.g., Upwork[2]) that enable the online collaboration of experts in order to complete projects, as well as the increasing popularity of online educational platforms (e.g., Coursera[3]) have brought team-formation problems into the spotlight, and have raised new questions and challenges.

The first work that addressed the problem of forming teams, taking into consideration not only the skills of the experts but also the communication between them, was the pioneering work of Lappas *et al.* [1]. In their setting, each worker is associated with a set of skills and there is also a network structure that captures how well a pair of workers can work together. The goal is to find a team that collectively covers the set of skills required for completing a task and it has low communication cost. Since the original work in [1], several extensions and variations of this general framework have been considered using different formulations for the communication cost, [2], [3], [4], [5], [1], [6], [13], [14], or different settings for job arrivals (e.g., offline vs. online) [7], [8].

Most of this existing work assumes a flat team structure, where all members are indistinguishable. However, in real life teams often have complex structures, and the team members have distinct roles within these structures [9]. For example, a team of authors writing a paper may consist of a professor, a post-doctoral fellow, and a few students. The post-doc acts as an intermediary in the communication of the students with the professor, while the students collaborate closely with each other to complete the experiments. Similarly, a team for completing a project in industry, may consist of a manager, a program manager, programming engineers, testing engineers, and researchers. These individuals are usually organized in a fixed hierarchy (different, for different corporations), and there are specific channels of communication between the different members of the team. Hierarchical structures, flat or tall, are very common in diverse types of organizations, such as corporations, governments, academia, or other institutions [9].

It is thus rarely the case that a team has a free-form structure, where anyone may assume any role, and everyone communicates with everyone. More often, there are specific roles to be filled, that fall within an organizational structure. Such cases are not addressed by the existing solutions. Motivated by these observations, we define the *Template-Driven Team Formation* (TDTF) problem, where given a fixed template structure for the team in the form of a graph, and a designated role for each node of the template, the problem asks for a team of workers that can fill the roles in the template, while minimizing the communication cost along the template edges. To the best of our knowledge, we are the first to formally define and study this problem. We show that although the TDTF problem is NP-hard in its full generality, there are variants of the problem for tree template graphs that an be solved optimally, using dynamic programming. For the hard variants, we design approximation and heuristic algorithms that exploit the properties of the problem.

We evaluate our algorithms experimentally on data from two different domains: academic collaborations, and collaborations in the movie-making industry. Our experiments demonstrate that our algorithms perform well in practice, while being quite efficient. We also conduct two case studies that confirm that the teams produced by our algorithms are highly intuitive.

[1]https://www.nytimes.com/2016/02/28/magazine/what-google-learned-from-its-quest-to-build-the-perfect-team.html

[2]http://www.upwork.com

[3]http://www.coursera.org

IEEE/ACM ASONAM 2020, December 7-10, 2020
978-1-7281-1056-1/20/$31.00 © 2020 IEEE

II. Related work

The problem of finding experts has attracted considerable attention in the research community [10], [11]. Balog et al., [12] were the first to consider the collaboration network in expert selection. This idea was formalized by Lappas et al., [1] who defined the team formation problem on social networks, where the goal is to identify a subset of experts with the required skills that induce a subgraph with low communication cost. A number of extensions and variations of this idea were considered in the follow-up work using different formulations for the communication cost [2], [3], [4], [5], [1], [6], or different settings for job arrivals [7], [8]. Recently a profit-maximization, load-balancing variant has been considered [13], [14], as well a game-theoretic view for strategic team formation [15]. None of these works considers teams with given structure and fixed roles. At their core, all these works involve solving an extended version of the set-cover problem, while our work corresponds to an assignment problem.

In this line of work, the work most closely related to ours is the work by Kargar and An [16]. In the problem variant they consider one of the team members is the designated leader and the rest of the team communicates mostly through the leader. Although this work tries to impose a structure on the identified teams, it only considers the simple star template, which is a special case of our problem, and it can be solved in polynomial time by an exhaustive algorithm.

The problem of having fixed roles in the team to which workers must be assigned is also considered in [17]. However, this work considers the problem of maximizing respect, rather than minimizing the communication cost, and it does not assume a fixed team structure.

A more recent line of work focuses on team-formation problems where the goal is for the formed teams to define a subnetwork with certain social-network properties (e.g., have certain number of dyads, triads, triangles, etc) [18]. There is a superficial similarity between this line of work and ours since in both cases there are some desired properties for the structure of the team network. However, the work in [18] focuses on optimizing some structural property of the network, while our work focuses on respecting the exact structure of an input template. The algorithmic techniques are also very distinct, since in [18] they use genetic algorithms, while we use combinatorial methods.

Finally, at a high level, our work is related to the work on graph pattern-matching [19], [20]. In this problem, we are given a graph pattern and we look for occurrences of the pattern in a graph, that is, subgraphs that are isomorphic to the pattern. We could view the template graph as a pattern that we want to find in the worker graph. However, this is a very restricted variant of our problem, where all teams have cost equal to the number of edges in the template. Our problem does not look for an exact match of the template graph, but rather the best possible assignment of workers to the template nodes.

III. Problem Definition

We are given a undirected connected graph $G = (W, E_G)$ which represents a network of workers. Each edge in E_G represents a connection between two workers, e.g., a past collaboration, or a personal relationship between the two workers. The edges may be weighted, where the weight denotes distance between the two workers. The shortest path distance $d(w, u)$ between two workers in the graph captures the degree of compatibility of the two workers. Small distance implies that the two workers can work well together.

Each worker has a set of skills. Given the set of all available skills, $\mathcal{S}$, we denote by $S_w \subseteq \mathcal{S}$ the set of skills of a worker $w \in W$. These skills may be programming languages if G represents a network of developers, or research fields if G represents a network of researchers. Every worker has at least one skill.

We want to create a team of workers for completing a task. We assume that we are given as input a template graph $T = (P, E_T)$. Each vertex p in the template represents a position in the team to be filled, and it is associated with a set of required skills $R_p \subseteq \mathcal{S}$. The structure of the template graph represents the communication structure in the team. For example if the template is a binary tree of depth two, we assume that the worker at the root of the tree communicates with her two subordinates, who in term communicate with their own two subordinates, and so on. It is thus important that there is good communication along the edges of the template graph. The goal is to fill the positions in the team, such that each worker has the required skills for the assigned position, and the workers assigned to neighboring positions have small distance, and thus can work together effectively.

We define a position assignment as a function $f : P \to W$, where worker $f(p)$ is assigned to position $p \in P$. The assignment f is *acceptable* if for every $p \in P$, $R_p \subseteq S_{f(p)}$, i.e., the worker assigned to the position has the required skills. We assume that the function f is *injective*, that is, a worker can only be used in a single position.

In order to evaluate an assignment f, we use the cost function $C(f)$ which is defined as the sum of the distances in G between the workers assigned to each pair of adjacent positions in T. Specifically,

$$C(f) = \sum_{(p,q) \in E_T} d(f(p), f(q))$$

We are now ready to define the *Template-Drive Team Formation* (TDTF) problem.

*Definition 1 (*TDTF*):* Given a network of workers $G = (W, E_G)$, with skills $\{S_w \subseteq \mathcal{S} : w \in W\}$, and a template $T = (P, E_T)$, with required skills $\{R_p : p \in P\}$ find an acceptable assignment $f : P \to W$, that minimizes the cost $C(f)$.

Given the general problem definition above, we can distinguish interesting subproblems by constraining the parameters of the problem. First, we constrain the number of skills a worker can have, i.e., the cardinality of the set S_w, for $w \in W$. We consider the special case where every worker has a single

TABLE I: Variants of the TDTF problem

	TDTF-SUT	TDTF-MUT	TDTF-SRT	TDTF-MRT	TDTF-SUG	TDTF-MUG	TDTF-SRG	TDTF-MRG
skills/worker	Single	Multiple	Single	Multiple	Single	Multiple	Single	Multiple
template skills	Unique	Unique	Repeated	Repeated	Unique	Unique	Repeated	Repeated
template graph	Tree	Tree	Tree	Tree	General	General	General	General

skill. We then constrain the number of times that a skill can appear in the template. We consider the special case where each position requires a unique set of skills, that is, each skill appears only once in the template. For simplicity, we assume a single skill per position in this case. Finally, we constrain the type of the template. We consider specific families of graphs that make sense in our setting. We will study in detail tree template graphs, which model hierarchical team structures, commonly found in real-life teams [9].

In order to differentiate between the different problem variants we will append a letter to the problem name that determines the variant we consider. We use the letters S and M to discriminate between **S**ingle and **M**ultiple skills per worker. We use the letters U and R to discriminate between **U**nique and **R**epeated skills in the template positions. We use the letter T to denote the **T**ree structure, and G to denote a **G**eneral graph. So, for example, the problem where we have a tree template graph, a single skill per worker, and unique skills in the template is denoted as TDTF-SUT. The general problem corresponds to the TDTF-MRG problem. In Table I, we provide a summary of the notation and the properties of the different problem variants.

We prove the following theorem for the problem complexity.

Theorem 1: All variants of the TDTF problem are NP-hard, except for the TDTF-SUT problem.

We omit the proof due to space constraints. In the next Section we show that there is a polynomial-time dynamic programming algorithm for the TDTF-SUT problem. This is the only variant of the problem that has a polynomial time solution. It is the combination of *all* three constraints that makes the problem tractable.

IV. Algorithms

We now present our algorithms for the TDTF problem. First, we show that the general problem can be solved in polynomial time in the case of star template graphs. Then, we consider the TDTF-SUT problem, and we show that there is a dynamic programming algorithm that solves the problem optimally. We then consider other variants of the problem on trees, and we propose a heuristic modification of the dynamic programming algorithm for these cases. Finally, we propose an algorithm for general template graphs, and we study the approximation guarantees for certain template graph families.

A. Algorithm for star templates.

The star template graph is a simple, yet natural template for teams, where we assume that there is "leader" to whom everyone reports. A similar problem has been considered in [16]. The TDTF problem in this case can be solved optimally with an algorithm that considers all possible candidate workers for the center of the star. For a position p, let W_p denote the workers in W that are candidates for this position, that is, they have the set of skills R_p. Let c denote the center of the star. The algorithm considers all candidate workers in W_c as possible assignments for the center. For a given assignment $f(c) = w$, $w \in W_c$, let p be a child of the center node c in the template. For the unassigned worker $u \in W_p$ the cost of assigning u to p is the distance $d(w, u)$, and it is independent of all other assignments to the leaves of the star. Each worker is candidate for multiple positions, and each position has multiple candidates. Finding the assignment with the minimum cost corresponds to finding a matching between positions and workers with the minimum cost. This can be solved in polynomial time using the Hungarian algorithm [21].

B. Dynamic Programming Algorithm for TDTF-SUT.

Recall that in the TDTF-SUT problem we assume a tree graph template, a single skill per worker, and unique skills in the template. We will show that this case can be solved optimally using a Dynamic Programming (*DP*) algorithm. The algorithm traverses the template tree structure in a bottom-up fashion, solving the problem for the subtrees, and then aggregating the solutions of the subproblems to solve bigger ones, until reaching the root of the tree.

Given a tree template $T = (P, E_T)$, we assume that the tree is rooted, and we use r to denote the root of the tree. For any node p in T we use T_p to denote the subtree rooted at node p. Let $\mathcal{F}(T_p)$ denote the set of all possible worker assignment functions for the subtree T_p. Let $\mathcal{F}_{w|p}(T_p)$ denote the set of all possible worker assignments where node p is assigned worker w. Let $f^*_{w|p} = \arg\min_{f \in \mathcal{F}_{w|p}} C(f)$ denote the assignment in $\mathcal{F}_{w|p}$ with the minimum cost. We use $B(w, T_p) = C(f^*_{w|p})$ to denote the cost of this assignment. For the overall optimal assignment f^*, we have $C(f^*) = \min_{w \in W_r} B(w, T)$ (recall that W_r is the set of candidate workers for position r). Also, if $w^* = \arg\min_{w \in W_r} B(w, T)$, then $f^* = f^*_{w^*|r}$.

The $B(w, T_p)$ values are computed recursively on the height of the tree as follows. For a subtree T_p of height zero, that is, a single leaf node in the template tree, we define $B(w, T_p) = 0$, since there is no communication cost involved. For a subtree T_p of height greater than zero, the cost of the solution that assigns worker w to the root of the tree p can be decomposed into the communication cost of worker w with the workers assigned to the children of the root, plus the cost for each subtree assignment. For each child q of the root p, we need to consider all candidate workers $x \in W_q$, and find the one that minimizes the sum $d(w, x) + B(x, T_q)$. The key observation is that since each worker has a single skill, and skills are

unique in the template, the sets of candidate workers for each position are disjoint, and thus we can consider each child independently. Therefore, letting $\mathrm{Chld}(p)$ denote children of node p in the template graph, we have:

$$B(w, T_p) = \sum_{q \in \mathrm{Chld}(p)} \min_{x \in W_q} \{d(w, x) + B(x, T_q)\} \quad (1)$$

Given the discussion above, we can now design a dynamic programming algorithm, *DP*, for finding the optimal assignment f^*. The algorithm maintains two $|W| \times |P|$ matrices B and F, where $B[w, p]$ stores $B(w, T_p)$, and $F[w, p]$ stores the optimal assignment $f^*_{w|p}$. The assignment is stored as a set of pairs $\{(w, q) : w \in W, q \in T_p\}$ that define the assignment of workers to positions.

The *DP* algorithm traverses the tree T in a post-order fashion, staring from the leaves and working its way up to the root r. Using Equation 1, at each node p it computes the value $B(w, T_p)$ and stores it in $B[w, p]$ along with the respective assignment $f^*_{w|p}$ in $F[w, p]$. When reaching the root of the tree r, it computes $w^* = \arg\min_{w \in W_r} B[w, r]$ and returns the assignment $F[w^*, r]$.

The complexity of the algorithm is determined by the sizes of the candidate sets of the skills in the template. For an edge (p, q) in the template we need to consider all pairs of candidates $|W_p| \times |W_q|$. If N_T is the size of the template, and N_s the popularity of the most popular skill, then the cost of the *DP* is $O(N_T N_s^2)$.

C. Heuristic Algorithms for Tree Templates

We now consider the TDTF-SRT, TDTF-MUT and TDTF-MRT variants of the problem, were we still have a tree template graph, but workers may have multiple skills, or the same skill may be repeated in the template. The common characteristic of these variants is that they allow a worker w to be candidate for more than one positions in the template. The *DP* algorithm we defined for the TDTF-SUT problem breaks down in this case, since the subproblems defined by the subtrees of a node are no longer independent. For example, a worker w that is eligible for two positions, may be the best candidate for both of these positions. As a result, w may appear in the optimal assignments for two subtrees T_p and T_q, which are children of a node v. Since we cannot assign worker w to both positions, it is no longer the case that we can express the cost for node v as a function of the optimal costs for nodes p and q.

We now consider heuristic algorithms for these problems.

1) Dynamic Programming Heuristic Algorithm (DPH): The first heuristic algorithm modifies the *DP* algorithm, addressing the issue of workers that are eligible for multiple positions in a greedy fashion. More specifically, when computing the cost $B(w, T_p)$ for the subtree rooted at position p, with w being assigned to the root, the algorithm maintains a set X_{wp} with all the workers that have already been assigned to some node in the subtree T_p. The algorithm iterates over the nodes in $\mathrm{Chld}(p)$ in an arbitrary order. When considering a position $q \in \mathrm{Chld}(p)$, it goes through the candidates $z \in W_q$ in decreasing order of the cost $d(w, z) + B(z, T_q)$. For a candidate $z \in W_q$, we have the set X_{zq} of all the workers that are utilized in the assignment $f^*_{z|q}$. If $X_{zp} \cap X_{wp} = \emptyset$, that is, if none of the workers in $f^*_{z|q}$ have already been used, then we add $f^*_{z|q}$ to the solution $f^*_{w|p}$, update the set X_{wp} accordingly, and move on to the next child of p. Otherwise, we discard this candidate, and move to the next one. If for some child of p there is no acceptable candidate, then we consider w as unacceptable for p, and move on to the next candidate for p. If no candidate for p produces a solution, then the algorithm halts and outputs no solution. Similar to before, the algorithm proceeds in a bottom-up fashion, until it reaches the root, or until it halts unable to produce a solution. The greedy aspect of the algorithm is that once it finds the lowest-cost assignment for a subtree (for a given root assignment) it discards all other assignments. However, as we move up the tree, this may result in excluding some assignments for sibling nodes that could yield a lower-cost solution overall. The pseudocode for the algorithm appears in Algorithm 1.

The complexity of the *DPH* algorithm is $O(N_T^2 N_s^2)$. The additional N_T factor comes from time needed to compute the intersection between the sets of assigned workers.

2) Top-Down Heuristics: We also consider two greedy heuristic algorithms that fill the template in a top-down fashion.

The *TopDown* algorithm: *TopDown* assigns to the root of the tree the worker with the highest closeness centrality, among the candidate workers. The closeness centrality for a worker in the graph G is defined as the inverse of the average distance of the node to all other nodes in the graph. Given the assignment for the root, the algorithm goes down the tree, each time assigning to a position the unassigned candidate worker that is closest to the worker of the parent node, until the whole template is filled.

The *TopDown+* algorithm: *TopDown+* is the same as *TopDown*, except for the fact that for the root assignment it considers all possible candidate workers in W_r. It then returns the assignment with the minimum cost.

The *MaxCentrality* baseline: We also consider a simple baseline that selects workers based on their closeness centrality in the network. The algorithm fills the positions in a top-down fashion, where for each position it selects the worker with the maximum centrality among the unused workers that have the required skill. This is a very efficient but naive algorithm, and we use it as a baseline in our experiments.

D. Algorithm for general templates

We now consider an algorithm for the TDTF problem on general templates. The algorithm exploits the fact that we have a methodology to solve the problem on trees. It first constructs a spanning tree of the template, by making a BFS traversal of the template graph. It then solves the TDTF problem using the BFS tree as the template, and computes the cost of the solution on the full template graph. The starting node for the BFS traversal determines the root and the structure of

Algorithm 1 Dynamic Programming Heuristic Algorithm (*DPH*)

Input: Graph $G = (W, E_G)$, template $T = (P, E_T)$, distance function d on graph G.
Output: optimal assignment f^*

$O \leftarrow PostOrder(P)$
$B \leftarrow |W| \times |P|$ Array storing $B(w, T_p)$
$F \leftarrow |W| \times |P|$ Array storing $f^*_{w|p}$
$X \leftarrow |W| \times |P|$ Array storing the workers in $f^*_{w|p}$
for $p \in O$ **do**
  for $w \in W_p$ **do**
    $B[w, p] \leftarrow 0$
    $F[w, p] \leftarrow \{(w, p)\}$
    $X[w, p] = \{w\}$
    for $q \in \text{Children}(p)$ **do**
      $mincost_q \leftarrow \infty$
      $L_q \leftarrow \{z \in W_q\}$ in decr. order of $d(z, w) + B[z, q]$
      for $z \in L_q$ **do**
        if $X[z, q] \cap X[w, p] = \emptyset$ **then**
          $mincost_q \leftarrow d(w, z)B[z, q]$
          $F[w, p] \leftarrow F[w, p] \cup F[z, q]$
          $X[w, p] \leftarrow X[w, p] \cup X[z, q]$
          $B[w, p] = B[w, p] + \text{mincost}_q$
          break
        end if
      end for
      if $mincost_q = \infty$ **then**
        $B[w, p] = \infty$
        break
      end if
    end for
  end for
  if $\min_{w \in W_p} B[w, p] = \infty$ **then**
    halt
  end if
end for
$w^* = \arg\min_{w \in W_r} B[w, r]$
return $F[w^*, r]$

the tree. For some template graphs the choice of the starting node is obvious. In the general case, the algorithm considers all possible starting nodes, and reports the solution with the minimum cost. We refer to this algorithm as the *Spanning Tree Algorithm (STA)*.

Despite the simplicity of the *STA* algorithm we can prove some interesting properties for it, by exploiting the triangular inequality of graph distances, and the fact that we have an optimal solution for the TDTF-SUT problem. For the following, let n denote the number of nodes, and m the number of edges in the template graph T. We prove the following Lemma for the TDTF-SUG problem, where we assume a general graph template, single skill per worker, and unique skills in the template. We omit the proof due to space constraints.

Lemma 1: The *STA* algorithm is a $(m-n+2)$-approximation algorithm for the TDTF-SUG problem.

The approximation bound in Lemma 1 is very loose for large m and n. However, note that m and n correspond to the edges and nodes in the template graph, which we expect to be small. Also, the bound in Lemma 1 is pessimistic, since for every edge of the template not in the spanning tree we charge the cost of the whole BFS tree. We can obtain better bounds for specific families of graphs.

Dataset	Skills
Academic	AI, Architecture, Computer Graphics, Data, Distributed-Parallel, HCI, Languages, Networks, OS, Security, Theory
Movies	Actor, Actress, Casting, Producer, Writer, Visual Effects, Director, Editor, Dir. of Photography, Screenplay, Art Director

TABLE II: Skills for *Academic* and *Movies* datasets

V. Experiments

The goals of the experiments are the following: (a) Compare the performance of different algorithms for the TDTF-SUT problem with respect to the cost metric, and study the effectiveness-efficiency tradeoff; (b) Study the performance of the heuristic algorithms for tree templates when workers can have multiple skills, and skills may appear in multiple positions (TDTF-MRT); (c) Compare the heuristic and approximation algorithms with an exhaustive algorithm on general graph templates; (d) Perform an empirical evaluation of the quality and intuitiveness of the teams produced by our algorithms by considering specific case studies.

A. Datasets

The *Academic* dataset: This dataset consists of information about academic publications, collected from Microsoft Academic[4]. We consider 11 fields of Computer Science, shown in Table II, and the corresponding conferences, shown in Table III. We collected all publications in the interval between 2000 and 2017 for these conferences, and the authors of these publications. We filtered out the authors with less than 7 publications in this interval, and we created the author collaboration graph, where each node is an author and there is an edge between two nodes if they have collaborated at least thrice in the specified time interval. We keep the largest connected component of this graph, resulting in a graph with 2,476 nodes and 3,853 edges. Each author is assigned as skills the fields in which she has authored a publication. When a single skill is required, we assign to each author the field in which she has the most publications.

The *Movies* dataset: This dataset consists of data about movies obtained from The Movie DB (TMDb)[5]. For the 3,000 most popular movies (according to TMDb) released after

[4]http://academic.microsoft.com
[5]https://www.themoviedb.org/

TABLE III: Fields and conferences in *Academic* dataset

Fields	Conferences
Theory	stoc, focs, soda, icalp, stacs
Languages	popl, icfp, icse, pldi, icsm
Distrib. & Parallel	podc, icdcs, spaa, ics, sc
Operating Systems	sosp, osdi, atc, fast, eurosys
Architecture	asplos, icsa, ispd, ches, iccd
Networks	sigcomm, nsdi, mobicom, mobisys, infocom
Security	usenix, oakland, crypto, acns, ccs
Data	sigmod, vldb, pods, kdd, www
Artificial Intelligence	aaai, icml, iccv, cvpr, acl
Computer Graphics	siggraph, i3d, mm, dcc, icme
H.C.I.	chi, cscw, uist, iui, gi

TABLE IV: Skill distributions and graph statistics.

Movies			*Academic*		
Skill	Single	Mult.	Skill	Single	Mult.
Actor	1020	1318	theory	186	300
Actress	1199	1323	distr-paral	58	411
Casting	487	503	ai	572	1099
Producer	1919	2433	os	25	195
Writer	609	1327	cg	442	920
Visual Effects	92	100	languages	21	111
Director	1151	1485	arch	11	100
Editor	699	830	security	44	157
Dir. of Photo.	545	588	hci	380	647
Screenplay	726	1121	data	420	901
Art Direction	708	730	networks	317	527
avg skills/worker	1	1.1758		1	2.168
avg workers/skill	832.27	1069		225.09	488
Total nodes	9155			2476	
Total edges	78832			3,853	

2010, we collected information about the cast and the crew of the movies. Given that the cast of a movie may contain tens of actors and actresses, we kept the 2,000 actors and 2,000 actresses that were on average ranked as most popular (according to TMDb) in our data. We also selected the crew members with key roles, shown in Table II, resulting in total 11 distinct roles. We created a graph by creating a node for each crew or cast member, and an edge between two nodes if they have collaborated in at least one movie. We subsampled 10K nodes to make the data manageable, and kept the largest connected component. The resulting graph has 9,155 nodes and 78,832 edges. Each node is assigned as skills all the roles she has assumed in the dataset. When a single skill is required, the most popular role is used.

Details and statistics for both datasets appear in Table IV.

B. *Results for* TDTF-SUT

Recall that for the TDTF-SUT problem we assume that the template is a tree, each worker has a single skill, and the skills appear in at most one position in the template. We construct the input for our experiments as follows. First, we assume that the template graph is a complete binary tree (CBT) of size ranging from 2 to 11 nodes (total number of skills), and there is a single skill per position. We start with a template of size 2, with 2 randomly chosen skills, and we construct templates of larger size, by adding one node at the time, with a skill randomly selected among the ones that have not already been used. In this way, we guarantee that the template of size n is a subset of the template of size $n + 1$. In our results we report the average performance of the algorithms over 50 such experiments.

Figure 1 shows the solution costs and the running times for our algorithms in the two datasets as a function of the template size. As expected, *DP* yields the lowest cost, at the expense of a much higher running time. The *MaxCentrality* baseline is significantly faster than all other algorithms, but with much higher solution cost. Between the two top-down heuristics, the *TopDown+* algorithm strikes the best balance between *DP* and *MaxCentrality*. Its running time is slightly higher than that of *TopDown* but it is still reasonably low, while the solution cost is almost identical to that of the optimal *DP* algorithm.

C. *Results for* TDTF-MRT

We now consider the TDTF-MRT problem, where the template graph is still a tree, but the same skill may appear multiple times in the template, and workers may have more than one skill. We generate the templates as before, but now, when assigning a skill to a position, we sample from all possible skills. We will study the performance of the different heuristics, and also consider templates of larger size.

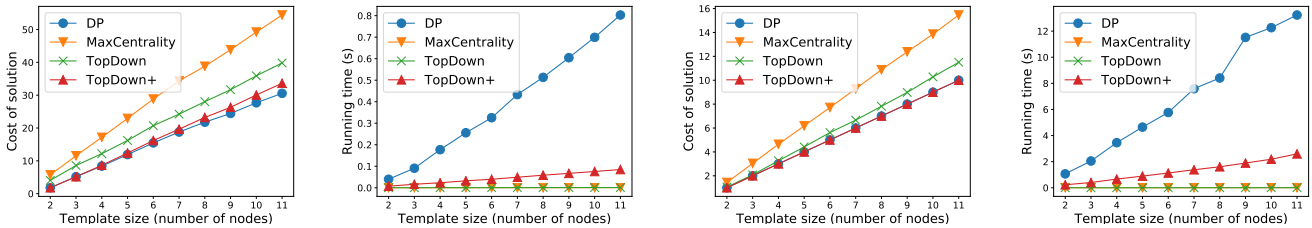
Figure 2 shows the results of our experiment. We observe again that the *DPH* algorithm performs best in terms of solution cost but has also the highest running time. The *TopDown+* algorithm is again the best option with low running time, and solution cost essentially identical to that of *DPH*.

Note also that the *DPH* algorithm may not always produce a solution. In our experiments, this was never the case for the *Movies* dataset, but for the *Academic* dataset, for template size between 26 to 31 we had failures ranging from 2% to 16%. These are non-negligible percentages, which demonstrate the weakness of *DPH* for large templates.

D. *Results for general graph templates*

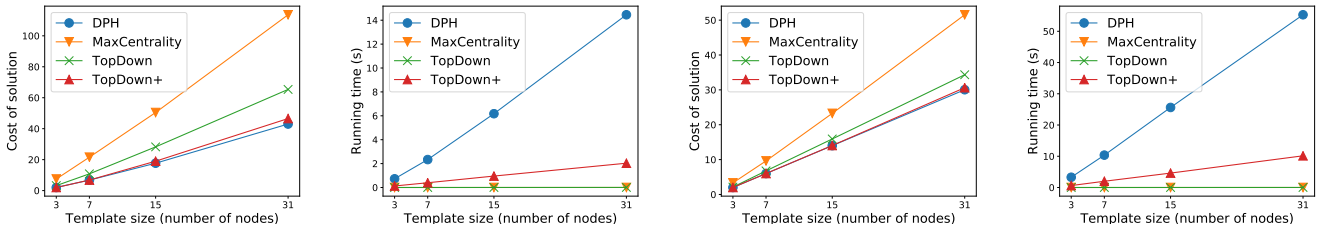
We now consider the TDTF problem on templates different from trees. Our goal is to study the performance of the *STA* algorithm against an exhaustive algorithm that considers all possible assignments.

Since it is computationally prohibitive to run the exhaustive algorithm on the full dataset, we construct smaller instances, by considering the *ego-network* of certain nodes in the network. The ego-network of a node consists of all the neighbors of the selected node, and all the edges between them. From the *Academic* dataset, we extracted the ego-network of Jon Kleinberg which consists of 34 nodes. We will refer to this dataset as *EgoKleinberg*. From the *Movies* dataset, we extracted the ego-network of George Clooney, which consists of 81 nodes. We will refer to this dataset as *EgoClooney*. To reduce the running time of the exhaustive algorithm we assume a single skill per worker. Also, since the neighbors in the *EgoKleinberg* network were heavily concentrated in just 3 fields, for this dataset we use the conferences as the skills.



(a) *Academic* solution cost (b) *Academic* running time (c) *Movies* solution cost (d) *Movies* running time

Fig. 1: Solution cost and running time for the TDTF-SUT problem with a CBT template.



(a) *Academic* solution cost (b) *Academic* running time (c) *Movies* solution cost (d) *Movies* running time

Fig. 2: Solution cost and running time for the TDTF-MRT problem with CBT template for varying template size.

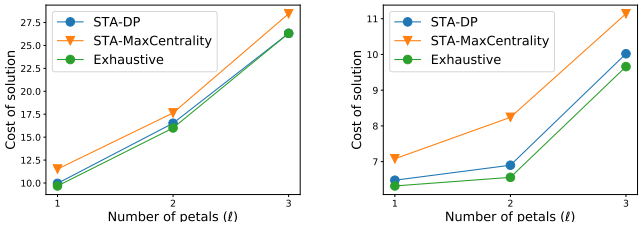
We considered a "flower" template for our experiments. In the flower template we have a center node that is connected to ℓk other nodes, which are organized in ℓ cliques ("petals") of size k. This corresponds to a structure with a single manager that manages ℓ teams of k workers that collaborate fully with each other. We can show that the *STA* algorithm has a k-approximation ratio for this family of templates. We set $k = 2$ and we vary ℓ from 1 to 3. For each template we conducted 50 different experiments with random skill assignments, where skills may be repeated in the template. We report the average cost of the solutions.

Figure 3 shows the results for the two datasets. We consider two variants of the *STA* algorithm, one that uses *MaxCentrality* to solve the problem on the spanning tree (*STA-MaxCentrality*), and one that uses *DPH* (*STA-DPH*). Note that the spanning tree is a star, so the solution of *DPH* and *TopDown+* on the spanning tree is optimal. We observe that the *STA-DPH* algorithm outperforms *STA-MaxCentrality*, and it is very close to that of the exhaustive algorithm, indicating that *STA* works well in practice.

E. Case Studies

Finally, we perform two case studies, one for each dataset. We consider the TDTF-SUT problem, and we manually set the template skills and evaluate the results. Our goal is to empirically evaluate the solutions of the *DP* algorithm.

For the *Academic* dataset, in order to make the experiment more interesting, we introduced an additional attribute to each researcher, which measures the seniority of the researcher. To this end, we used the total citation count of each author, as provided by Microsoft Academic, which we mapped to the nominal values *senior*, *middle*, and *junior*. We label



(a) *EgoKleinberg* (b) *EgoClooney*

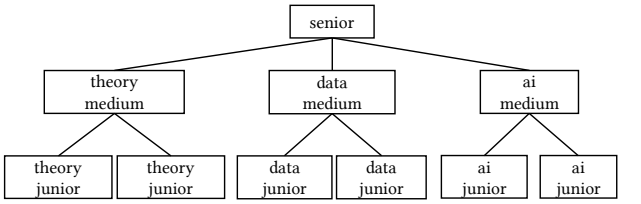
Fig. 3: Solution costs for flower graph templates, for varying number of petals (ℓ) with fixed petal size $k = 2$.

researchers with citations in the top-5% (more than 17,165 citations) as *senior*, researchers in the top-35% (more than 3,752 citations) as *middle*, and the rest as *junior*[6]. Using the seniority attribute, we construct skills that use a combination of the seniority and a research field.

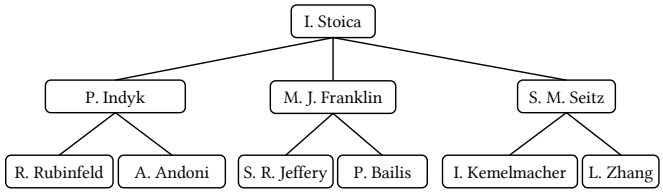
The template we used in our experiment with *Academic* is shown in Figure 4a. The scenario we consider is that of creating a new research lab. The head of the lab should be a senior researcher, irrespective of the field. There are three divisions, one in Theory one in Data, and one in AI, which will be headed by a researcher of middle seniority in the field. Each division head will manage two junior researchers in their respective field.

The result we obtain, shown in Figure 4b, is highly intuitive. Ion Stoica, Professor at U.C. Berkeley, authority in the field of distributed systems with a broad set of interests, is the

[6]Some seniority labels are debatable. The problem of defining the right notion of seniority is beyond the scope of our paper.

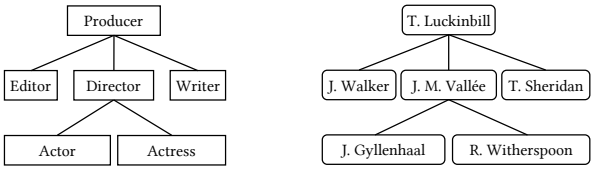


(a) Template & Skills



(b) *DP* Solution

Fig. 4: *Academic* case study.



(a) Template & Skills (b) *DP* Solution

Fig. 5: *Movies* case study.

head of the lab. Piotr Indyk, Professor at MIT, authority in theoretical computer science, is the head of the Theory division. P. Indyk has common collaborators with I. Stoica (Sammuel Madden). He manages his former student, Alexandr Adoni, and Ronitt Rubinfeld who is also professor at MIT. Hence both are academically close to him. The head of the Data division is Michael Franklin, longtime collaborator of I. Stoica, highly respected in the field of Data Bases. He manages his former Ph.D. student Shawn R. Jeffery, and Peter Bailis, U.C. Berkeley graduate and former Ph.D. student of I. Stoica, with whom he has co-authored several papers. The head of the AI division is Steven Seitz, an expert in computer vision. He received his Bachelor from U.C. Berkeley, and he has common collaborators with I. Stoica (e.g., Sameer Agarwal). He manages two of his former Ph.D. students, Ira Kemelmacher-Shilzerman and Li Zhang.

The template we use for the *Movies* dataset is shown in Figure 5a. We have a Producer at the root of the tree who employs an Editor, a Director and a Writer, and the Director collaborates with an Actor and an Actress. The solution of the *DP* algorithm is shown in Figure 5b. The Producer, T. Luckinbill, has worked together with T. Sheridan and J. Walker in the movie Sicario (2015) and with J. M. Vallee in Demolition (2015) in which J. Gyllenhaal stars as a lead actor. Also, R. Witherspoon stars in Vallee's movie Wild (2014). Therefore, again the solution we obtain is highly intuitive.

VI. Conclusion

In this paper we introduced and studied the novel problem of template-driven team formation (TDTF). We showed that the problem is NP-hard in the general case, but it can be solved in polynomial time using dynamic programming for tree templates when workers have a single skill, and the template positions have unique skills. We provided heuristic and approximation algorithms for the general case. Our experiments demonstrate that our algorithms are effective in practice. For future work, we are interested in studying the effect of the template graph on the team formation problem and on the problem approximability, and also incorporate ideas from skill-specific ranking [22] into the team formation process.

References

[1] T. Lappas, K. Liu, and E. Terzi, "Finding a team of experts in social networks," in *ACM SIGKDD*, 2009.

[2] R. Bredereck, J. Chen, F. Hüffner, and S. Kratsch, "Parameterized complexity of team formation in social networks," *Theoretical Computer Science*, vol. 717, pp. 26 – 36, 2018.

[3] A. Bhowmik, V. S. Borkar, D. Garg, and M. Pallan, "Submodularity in Team Formation Problem," in *SIAM SDM*, 2014.

[4] A. Gajewar and A. D. Sarma, "Multi-skill Collaborative Teams based on Densest Subgraphs," in *SDM*, 2012.

[5] B. Golshan, T. Lappas, and E. Terzi, "Profit-maximizing cluster hires," in *ACM SIGKDD*, 2014.

[6] S. S. Rangapuram, T. Bühler, and M. Hein, "Towards Realistic Team Formation in Social Networks Based on Densest Subgraphs," in *WWW*, 2013.

[7] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi, "Power in unity: forming teams in large-scale community systems," in *ACM CIKM*, 2010.

[8] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi, "Online team formation in social networks," in *WWW*, 2012.

[9] Rishipal, "Analytical comparison of flat and vertical organizational structures," *European Journal of Business and Management*, vol. 6, no. 36, 2014.

[10] C. S. Campbell, P. P. Maglio, A. Cozzi, and B. Dom, "Expertise identification using email communications," in *ACM CIKM*, 2003.

[11] J. Zhang, M. S. Ackerman, and L. Adamic, "Expertise networks in online communities: Structure and algorithms," in *ACM WWW*, 2007.

[12] K. Balog, L. Azzopardi, and M. de Rijke, "Formal models for expert finding in enterprise corpora," in *ACM SIGIR*, pp. 43–50, 2006.

[13] S. Tang, "Profit-driven team grouping in social networks," in *AAAI Conference on Artificial Intelligence*, p. 45–51, AAAI Press, 2017.

[14] S. Liu and C. K. Poon, "A simple greedy algorithm for the profit-aware social team formation problem," in *Combinatorial Optimization and Applications* (X. Gao, H. Du, and M. Han, eds.), (Cham), pp. 379–393, Springer International Publishing, 2017.

[15] W. Wang, Z. He, P. Shi, W. Wu, Y. Jiang, B. An, Z. Hao, and B. Chen, "Strategic social team crowdsourcing: Forming a team of truthful workers for crowdsourcing in social networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 6, pp. 1419–1432, 2019.

[16] M. Kargar and A. An, "Discovering Top-k Teams of Experts with/Without a Leader in Social Networks," in *ACM CIKM*, 2011.

[17] S. M. Nikolakaki, E. Pitoura, E. Terzi, and P. Tsaparas, "Finding teams of maximum mutual respect," in *ICDM*, IEEE, 2020.

[18] A. Farasat and A. G. Nikolaev, "Social structure optimization in team formation," *Computers & OR*, vol. 74, pp. 127–142, 2016.

[19] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu, "Graph pattern matching: From intractable to polynomial time," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 264–275, 2010.

[20] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE TPAMI*, 2004.

[21] B. Korte and J. Vygen, *Weighted Matching*, pp. 235–260. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002.

[22] M. Coscia, G. Rossetti, D. Pennacchioli, D. Ceccarelli, and F. Giannotti, ""You know because i know": A multidimensional network approach to human resources problem," in *ASONAM*, 2013.