Template-driven team formation

A Thesis

submitted to the designated
by the General Assembly of Special Composition
of the Department of Computer Science and Engineering
Examination Committee

by

Spiros Apostolou

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WITH SPECIALIZATION
IN SOFTWARE

University of Ioannina
February 2018

Examining Committee:

- **Panayiotis Tsaparas**, Associate Professor, Computer Science & Engineering Department, University of Ioannina (Supervisor)
- **Evaggelia Pitoura**, Professor, Computer Science & Engineering Department, University of Ioannina
- **Nikos Mamoulis**, Associate Professor, Computer Science & Engineering Department, University of Ioannina

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Spiros Apostolou, M.Sc. in Computer Science, Department of Computer Science and Engineering, University of Ioannina, Greece, February 2018.
Template-driven team formation.
Advisor: Panayiotis Tsaparas, Associate Professor.

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. This is a problem of great practical importance that has attracted considerable attention. Most related work assumes a flat structure in the team, where team members are all indistinguishable, or a simple star structure centered around a leader. However, in real life, teams often have complex structures and deep hierarchies, and members with distinct roles in these structures. In this thesis, 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.

Εκτεταμένη Περίληψη

Σπύρος Αποστόλου, Μ.Δ.Ε. στην Πληροφορική, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Φεβρουάριος 2018.
Σχηματισμός ομάδος με χρήση προσχεδίου.
Επιβλέπων: Παναγιώτης Τσαπάρας, Αναπληρωτής Καθηγητής.

Το πρόβλημα του *σχηματισμού ομάδας*(team-formation) σε κοινωνικά δίκτυα αφορά στην αναζήτηση μίας ομάδας ατόμων που συνολικά έχει τις απαραίτητες ικανότητες (skills) για να φέρει εις πέρας μία εργασία, ενώ ταυτόχρονα το κόστος επικοινωνίας, το οποίο μετριέται ως το άθροισμα των αποστάσεων των μελών στο κοινωνικό δίκτυο, παραμένει σε χαμηλά επίπεδα. Η πρακτική φύση του προβλήματος είναι και ο λόγος που έχει προσελκύσει το ενδιαφέρον των ερευνητών σε μεγάλο βαθμό. Δεν είναι ασυνήθιστο σενάριο μια εταιρία να θέλει να σχηματίσει μια ομάδα για να διεκπεραιώσει μία εργασία ή ένα τμήμα πανεπιστημίου να θέλει να σχηματίσει μια ομάδα από ερευνητές για ένα πρόγραμμα το οποίο απαιτεί άτομα από διαφορετικούς τομείς της πληροφορικής. Η βιβλιογραφία, στην πλειοψηφία της, υποθέτει πως η ομάδα έχει επίπεδη δομή, όπου δηλαδή όλα τα μέλη είναι ίσα, ή μία απλή δομή αστεριού όπου όλα τα μέλη είναι συγκεντρωμένα γύρω από έναν ηγέτη. Παρ' όλα αυτά, στην πραγματική ζωή, οι ομάδες έχουν σύνθετες δομές και ιεραρχίες με μεγάλο βάθος και τα μέλη των ομάδων έχουν διακριτούς ρόλους. Σε αυτή την εργασία, ορίζουμε το πρόβλημα του *Σχηματισμού Ομάδος με Πρότυπο*, σύμφωνα με το οποίο δοθέντων ενός κοινωνικού γράφου που ενώνει, ενός προτύπου δομής της ομάδος σε μορφή γράφου και μιας ανάθεσης ρόλων στους κόμβους του προτύπου, αναζητούμε "εργάτες" οι οποίοι θα καταλάβουν τις θέσεις του προτύπου, ελαχιστοποιώντας το κόστος επικοινωνίας που ορίζεται ως το άθροισμα των αποστάσεων των ατόμων που επιλέχθηκαν για να καταλάβουν τις θέσεις του προτύπου ως προς τις ακμές του προτύπου, δηλαδή η απόσταση δύο ατόμων που δεν ενώνονται με ακμή στο

πρότυπο δεν συνυπολογίζεται στο κόστος της ανάθεσης καθώς η καλή επικοινωνία ανάμεσα τους είναι δευτερευόντος σημασίας αφού δεν προβλέπεται να χρειαστεί να συνεργαστούν. Παρ' όλο που το πρόβλημα είναι NP-hard, μερικές παραλλαγές του προβλήματος λύνονται βέλτιστα χρησιμοποιώντας δυναμικό προγραμματισμό. Για την γενική περίπτωση, παρέχουμε προσεγγιστικές λύσεις και ευριστικούς αλγόριθμους. Επιπλέον πειραματιζόμαστε σε πραγματικά δεδομένα και παρουσιάζουμε πως οι ευριστικοί αλγόριθμοι αποδίδουν εξίσου καλά ενώ ταυτόχρονα είναι πολύ πιο αποδοτικοί σε θέματα χρόνου εκτέλεσης, και πως οι αλγόριθμοι με τους οποίους επιδιώκουμε να προσεγγίσουμε την βέλτιστη λύση στην γενική περίπτωση δεν απέχουν πολύ από αυτή. Τέλος, διεξάγουμε μια εμπειρική μελέτη των αποτελεσμάτων η οποία επισημαίνει την ποιότητα αυτών, καθώς οι ομάδες που παράγονται για κάθε ομάδα δεδομένων είναι αληθοφανείς, ενώνει δηλαδή άτομα που είναι λογικό να ανήκουν στην ίδια ομάδα βάσει των προηγούμενων συνεργασιών τους.

Chapter 1

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[1]) 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[2]) 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.* [2]. 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 [2], several extensions of this general framework have been considered, using different formulations for the communication cost [3, 4, 5, 6, 2, 7], or different settings for job arrivals [8, 9].

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

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

Most of this existing work, assumes a flat team structure, where all members are indistinguishable, or they assume that there is one team leader to which everyone is connected [10]. However, in real life teams often have complex structures and hierarchies, and the team members have distinct roles within these structures. For example, in a team of authors writing a paper, there may be a senior professor that gives the direction for the paper and has the overall supervision, a post-doctoral researcher that goes into the technical details of the paper, and a few students that are in charge of running the experiments. 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 organizations), and there are specific channels of communication between the different members of the team.

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. From the technical point of view, we study the complexity of the problem, we provide optimal, approximation, and heuristic algorithms for the different variants of the problem, and we perform experiments on two datasets that demonstrate the effectiveness and efficiency of our algorithms.

1.1 Contributions

In summary, the contributions of this thesis are the following:

- We define the novel problem of *Template-Driven Team Formation* (TDTF). 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 when the template is a tree, that can be solved optimally, using dynamic programming. For the hard variants, we design ap-

proximation and heuristic algorithms that exploit the properties of the problem.

- We perform experiments 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.

1.2 Roadmap

The rest of the thesis is organized as follows. Chapter 2 reviews the related work in the area. In Chapter 3 we formally define our problem, and its different variants, and study their complexity. In Chapter 4 we present our algorithms for the different problem variants. Chapter 5 presents the experimental evaluation of our algorithms, and Chapter 6 concludes this thesis.

Chapter 2

Related work

The high-level goal in team-formation problems is the following: given a set of experts organized in a network, where each individual is associated with a set of skills, identify a subset of experts that together can perform a given task, while at the same time they induce a subgraph with low communication cost [8, 9, 3, 4, 5, 6, 2, 7]. At their core, all these problems involve solving an extended version of the set-cover problem. None of these works considers hierarchical teams, or teams described by a graph template with fixed roles. As a result the algorithmic problems are completely different from those considered in our work. A fundamental difference is that our problem does not correspond to a coverage problem, but rather an assignment problem.

Among the works that formalize the team-formation problem as a variant of the set-cover problem, the most related to ours is the work by Kargar and An [10]. In that work, they consider a variant of the team-formation problem where one of the team members is a designated leader and the rest of the team communicates mostly through the leader. Although this work tries to impose a structure in the identified teams, it only considers the simple star template, which, as we show, is a special case of our problem, and it can be easily solved by an exhaustive algorithm. The formulation in [10] does not generalize to more complex templates. As a result, both the high-level problem definition we consider, as well as the algorithmic challenges we face are quite different from the ones addressed in [10].

Going beyond set-cover formulations, other formulations for team formation have been recently considered in the context of teams being formed in educational set-

tings (both offline and online) [11, 12, 13, 14]. The work in this domain comes in two flavors. The first, focuses on finding one or multiple teams of students so that some learning objective is maximized [11, 12]. The second, focuses on the a-posteriori analysis of already formed teams (usually formed in an ad-hoc way). These studies identify factors that make a team successful or not [13, 14]. The objective function we optimize in our work is different from the ones studied in this line of work. Moreover, in the teams studied in educational settings no template is given as part of the input.

To some, the team formation problem may sound similar to the graph pattern matching problem, which seeks for a pattern (similar to this work's template) inside a graph [15, 16]. In pattern matching and related bibliography, the solution is expected to be isomorphic to the pattern, which in our setup means that every assignment should have cost $|E_T|$. This is a very strict variant to our problem and highly counter-intuitive since it actually looks for preconstructed teams instead of looking to form new ones. Some relaxations of the problem have been discussed [17, 18], but they don't really approach our problem since these relaxations only mean to improve the solutions while still pertaining to the strict variant where every team member must be a direct neighbor to each of his colleagues.

Finally, 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) [19]. One can draw some high-level parallels between this line of work and ours: in both cases there are some desired properties for the structure of the team network. However, this similarity is only at a very high level. The work of Farasat and Nikolaev [19] 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 used in the two papers are also very distinct. Farasat and Nikolaev use genetic algorithms, while we use combinatorial methods like dynamic programming.

Chapter 3

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 effectively together.

Each worker has a set of skills. Given the set of all available skills, $\mathcal{S}$, the set of skills of a worker $w \in W$ is denoted as $S_w \subseteq \mathcal{S}$. These skills may be programming languages if G represents a network of developers, or they may be a set of 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. 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

Table 3.1: Variants of the TDTF problem

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

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 3.1 (Template-Driven Team Formation (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. The first parameter we constrain is the number of skills a worker can have, i.e. the cardinality of the set S_w, for $w \in W$. We consider the case where every worker has a single skill. The second parameter we constrain is the number of times that a skill can appear in the template, and we consider the case where each position has a unique set of skills. Finally, the

third parameter we constrain is 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 the case of a hierarchical team structure which is commonly found in real-life teams.

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 problem where we have a tree template but no restriction on the number of skills per worker, or the number of appearances of skills in the template is denoted as TDTF-MRT. The general problem corresponds to the TDTF-MRG problem. We will also consider additional families of templates later on, and introduce the corresponding notation.

Table 3.1 summarizes the properties of the different problem variants we consider in detail in this thesis, and serves as a reference for the problem-naming notation.

We will now consider the computational complexity of our problem. We can prove the following theorem for the general TDTF problem.

Theorem 3.1. *The Template-Driven Team Formation (*TDTF*) problem is NP-hard.*

Proof. The decision version of the TDTF problem asks if there is an assignment $f : P \to W$ with cost $C(f) \leq \theta$, for some value θ. It is easy to show that we can reduce the SubgraphIsomorphism problem [20] to our problem. The SubgraphIsomorphism problem, given two graphs G and H as input, asks if graph G contains a subgraph isomorphic to H. We can reduce an instance of SubgraphIsomorphism to an instance of TDTF where G is the worker graph, and H the template graph. We set all workers to have the same skill s, and all positions to require the same skill s as well. Setting $\theta = |E_H|$ to be the number of edges in the graph H, it is easy to see that G contains an isomorphic subgraph H, if and only if there is an assignment f with cost $C(f) \leq \theta$.

Note that the same reduction works directly for the TDTF-SRG problem, while for the TDTF-MUG problem we can change the reduction to give a distinct skill to each node in the template, while the skill set of each worker consists of the set of all

possible skills in the template. Given that the subgraph isomorphism problem remains NP-hard when the graph H is a tree, it follows that the TDTF-MUT, TDTF-SRT or TDTF-MRT problems are also hard. □

We can also prove the following theorem for the TDTF-SUG problem.

Theorem 3.2. *The* TDTF-SUG *problem is NP-hard.*

Proof. We will prove the theorem using a reduction from the k-Clique problem on a k-partite graph which is known to be NP-hard [20]. Given a k-partite graph H, we reduce the k-Clique problem on H to the TDTF-SUG problem as follows. We define the worker graph G to be H. We define k skills and we assign the same skill to all nodes in the same partition. We define the template graph T to be a k-clique, and we assign a different skill to each position in the template. It is easy to see that there is a worker assignment f with cost $C(f) \leq \binom{k}{2}$ if and only if there is a k-clique in H. □

In Chapter 4 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.

Chapter 4

Algorithms

We now present our algorithms for the TDTF problem. First, we show that the general problem can be solved easily 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 show that it has a provable approximation factor for certain template graph families.

4.1 Algorithm for star templates

The star template graph is a simple, yet natural template for teams, where we assume that there is "leader" that has put the team together. A similar problem has been considered in [10]. The TDTF problem in this case can be solved optimally with an algorithm that simply 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 $f(c) = w$, $w \in W_c$, for every child p of the center node c in the template, we find the worker $u \in W_p$ that has not already being assigned to a position and minimizes the distance $d(w, u)$, and we set $f(p) = u$. It is not hard to see that the assignment with the minimum cost is optimal. Note that this algorithm works for any problem setting, including single and multiple worker skills, and unique and repeated skill appearances in the template.

4.2 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 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. If f^* is the optimal assignment overall, then clearly $C(f^*) = \min_{w \in W_r} B(w, T)$. Also, if $w^* = \arg\min_{w \in W_r} B(w, T)$, then $f^* = f^*_{w^*|r}$.

Given a network of workers $G = (W, E_G)$, the algorithm utilizes two $|W| \times |P|$ matrices M and F, where $M[w, p]$ stores $B(w, T_p)$, and $F[w, p]$ stores the optimal assignment $f^*_{w|p}$ of workers in W to positions in the subtree T_p rooted at node p, given that position p is filled by worker w. 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 $B(w, T_p)$ values are computed recursively on the height of the tree as follows. For a subtree T_p of height zero, that is, a 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 in the template, plus the cost for each subtree for these assignments. 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, we can consider each child independently. Therefore, letting Children(p) denote children of the root node p in the template graph, we have:

$$B(w, T_p) = \sum_{q \in \text{Children}(p)} \min_{x \in W_q} \{d(w, x) + B(x, T_q)\}, \tag{4.1}$$

where W_q denotes the set of candidate workers that have the skill in position q.

Given Equation 4.1, 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. At each node p it computes the value $B(w, T_p)$ and stores it in $M[w, p]$ alongside the respective assignment $f^*_{w|p}$ in $F[w, p]$. When reaching the root of the tree r, it computes $w^* = \arg\min_{w \in W_r} M[w, r]$ and returns the assignment $F[w^*, r]$. The pseudocode for the algorithm is shown in Algorithm 4.1.

Algorithm 4.1 Dynamic programming algorithm for TDTF-SUT

Input: Graph $G = (W, E_G)$, template $T = (P, E_T)$, distance function d

Output: optimal assignment f^*

$O \leftarrow PostOrder(W)$
$M \leftarrow$ new $N \times P$ Array
for $p \in O$ **do**
    for $v \in C_p$ **do**
        $sum_v \leftarrow 0$
        for $u \in desc(p)$ **do**
            $m_u \leftarrow \min_{w \in C_u} \{B(w, u) + d_G(v, w)\}$
            $sum_v \leftarrow sum_v + m_u$
        end for
        $M_{vp} \leftarrow sum_v$
    end for
end for

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 is popularity of the most popular skill, then the cost of the *DP* is $O(N_T N_s^2)$.

4.3 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 it is no longer the case that the subproblems defined by the subtrees of the root node are independent. For example, a worker w that is eligible for two positions, may be the best candidate for both of these positions. Assume that w appears in the optimal assignments for the subtrees T_p and T_q, which are children of 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.

4.3.1 Dynamic Programming Heuristic Algorithm (DPH)

The first heuristic algorithm modifies the *DP* algorithm, addressing the issue of workers that are eligible for multiple positions. More specifically, when computing the cost $B(w, T_p)$ for the subtree rooted at position p, when assigning w to the root, the algorithm iterates over the nodes in Children(p) in an arbitrary order. Throughout the iterations, it maintains a set X_{wp} with all the workers that have already been assigned to some node in the subtree T_p. When considering a position $q \in$ Children(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 pseudocode for the algorithm is shown in Algorithm 4.2.

4.3.2 Top-Down Heuristics

We also consider two greedy heuristic algorithms that fill the template in a top-down fashion. The *TopDown* algorithm assigns randomly one of the candidate workers to the root of the tree. For the children of the root, it assigns the candidate workers that are closest to the root worker, making sure that no worker is used twice. The algorithm proceeds like that, down the tree, each time assigning to a node the candidate worker that is closest to the worker of the parent node that has not already been used, until the whole template is filled.

The 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. Note that the *TopDown+* algorithm is optimal for the case of the star template.

4.4 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 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, exploiting the triangular inequality of graph distances. For the following,

Algorithm 4.2 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^*

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

let n denote the number of nodes in the template graph T, and m the number of edges. 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.

Lemma 4.1. *The STA algorithm is a* $(m - n + 2)$*-approximation algorithm for the* TDTF-SUG *problem.*

Proof. Let T be the input template graph, and let SP_T denote the spanning tree for the template T. Given SP_T as input we can solve the TDTF-SUT problem optimally using the *DP* algorithm. Let f denote the optimal assignment for the spanning tree produced by the *DP* algorithm. The cost of the assignment f on the template T is defined as:

$$\begin{aligned} C(f) &= \sum_{(p,q)\in T} d\left(f(p), f(q)\right) \\ &= \sum_{(p,q)\in SP_T} d\left(f(p), f(q)\right) + \sum_{(p,q)\notin SP_T} d\left(f(p), f(q)\right) \end{aligned}$$

For any edge $(p, q) \notin SP_T$ that is not in the spanning tree, let $\text{Path}(p, q)$ denote the path of edges in SP_T that connects the two vertices. Since the graph distance d satisfies the triangular inequality, we have that

$$\begin{aligned} d\left(f(p), f(q)\right) &\leq \sum_{(x,y)\in \text{Path}(p,q)} d\left(f(x), f(y)\right)) \\ &\leq \sum_{(p,q)\in SP_T} d\left(f(p), f(q)\right) \end{aligned}$$

There are $m - n + 1$ such edges, therefore, we have that

$$C(f) \leq (m - n + 2) \sum_{(p,q)\in T} d\left(f(p), f(q)\right)$$

Let f^* be the optimal assignment for the template graph T. Since f is optimal on the spanning tree SP_T it holds that

$$\sum_{(p,q)\in SP_T} d\left(f(p), f(q)\right) \leq \sum_{(p,q)\in SP_T} d\left(f^*(p), f^*(q)\right) \leq C(f^*)$$

It follows that $C(f) \leq (m - n + 2)C(f^*)$. □

As an immediate corollary, if we restrict the TDTF-SUG problem to Cycle graphs, the *STA* algorithm has a 2-approximation factor with respect to the optimal solution.

Note that the bound in Lemma 4.1 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 by bounding the length of the Path(p, q) in the spanning tree that connects the endpoints of the edge (p, q) in the template.

For example, in our experiments, we consider the family of "flower" graphs. A flower graph $F_{\ell,k}$ is a graph with $k\ell+1$ nodes. There is a center node that is connected to all other $k\ell$ nodes. The children of the center node are organized in ℓ cliques of size k. This corresponds to the case where we have a leader in the team, to whom everyone in the team reports. The subordinates of the leader are organized into ℓ teams of size k, where everyone works with every one.

Let TDTF-MRF denote the TDTF problem on flower graphs. We can prove the following.

Proposition 4.1. *The STA algorithm gives a k-approximation solution for the* TDTF-MRF *problem.*

Proof. We run the *STA* algorithm using the center node as the root of the BFS tree. The spanning tree in this case is a star, so we can solve the problem optimally, when we use it as input template. Let f denote this optimal assignment.

For every clique of size k, there are $\binom{k}{2}$ edges that are not included in the BFS tree. For every edge (p, q) there is a path $\{(p, c), (c, q)\}$ that connects them, where c is the center of the flower. Note that the edge (c, p) participates in $(k-1)$ such paths, for all the neighbors of p in the clique. Therefore, we have that:

$$\begin{aligned} C(f) &= \sum_{(c,p)\in SP_T} d\left(f(c), f(p)\right) + \sum_{(p,q)\notin SP_T} d\left(f(p), f(q)\right) \\ &= \sum_{(c,p)\in SP_T} d\left(f(c), f(p)\right) + (k-1) \sum_{(c,p)\in SP_T} d\left(f(c), f(p)\right) \\ &\leq kC(f^*) \end{aligned}$$

where f^* denotes the optimal solution. The last inequality follows from the fact that the solution on the star BFS tree is optimal, and the tree is a subset of the template. □

Chapter 5

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.

5.1 Datasets and Baselines

We now describe the two datasets we consider in this thesis, and a simple baseline algorithm we will use for comparisons.

5.1.1 The Academic dataset

This dataset consists of information about academic publications, collected from Microsoft Academic[1]. We consider 11 fields of Computer Science and the corresponding conferences, shown in Table 5.1. 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 collaboration graph between authors, where each node is an author and there is an edge connecting two nodes if they have collaborated at least thrice in the specified time interval. We keep the largest connected component of this graph. Each author is assigned as skills the fields in which she has authored a publication. For the problems where each worker should have a single skill, we assign to each author the field in which she has the most publications. The statistics for our dataset are shown in Table 5.2.

Table 5.1: Fields and conferences in *Academic* dataset

Fields	Conferences
Theory	stoc, focs, soda, icalp, stacs
Languages	popl, icfp, icse, pldi, icsm
Distributed & Parallel Computingx	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
Human-Computer Interaction	chi, cscw, uist, iui, gi

[1] http://academic.microsoft.com

5.1.2 The Movies dataset

This dataset consists of data about movies obtained from The Movie DB (TMDb)[2]. For the 3,000 most popular movies released after 2010, we collected information about the cast and the crew of the movies. For the movie popularity we used a value provided by TMDb, which is calculated by taking into account the movie's ratings, TMDb page views, release date, and more. Given that the cast of a movie may contain tens of actors and actresses, we initially kept the 2,000 actors and 2,000 actresses that were on average ranked as most popular (according to TMDb) in our data. From the crew, we selected the roles shown in Table 5.2, 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. 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. The statistics for the graph are shown in Table 5.2

5.1.3 The MaxCentrality baseline

We also consider a simple baseline that selects workers based on their closeness centrality in the network. 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. 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 clearly, a very efficient but naive algorithm, and we use it as a baseline for the efficiency and effectiveness of the algorithms we introduced in Section 4.

5.2 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

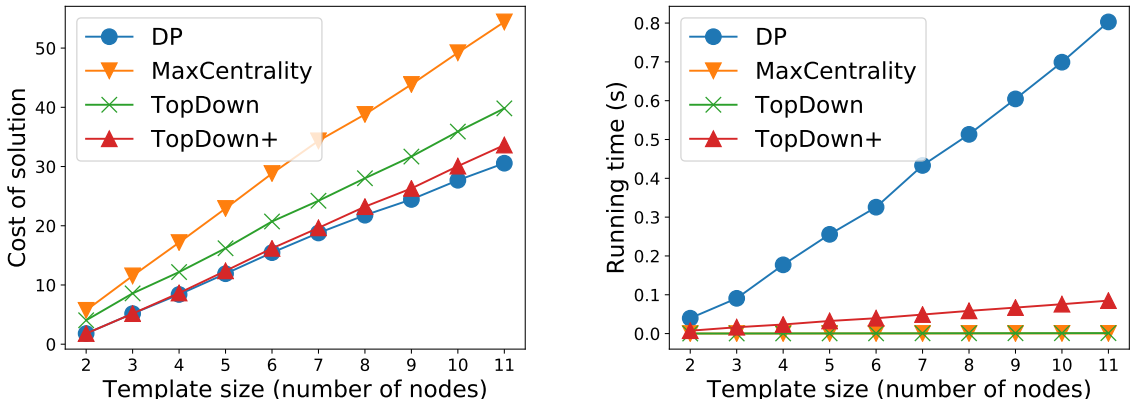
[2]https://www.themoviedb.org/

Table 5.2: Skill distributions and graph statistics.

Movies			*Academic*		
Skill	Single	Multiple	Skill	Single	Multiple
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			9155	

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 construct the template incrementally. 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.

Figures 5.1 and 5.2 show 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 with much higher solution cost. Between the two top-down heuristics, the *TopDown+* algorithm strikes the best



(a) *Academic* solution cost

(b) *Academic* running time

Figure 5.1: Solution cost and running time for the TDTF-SUT problem with a CBT template for the *Academic* dataset.

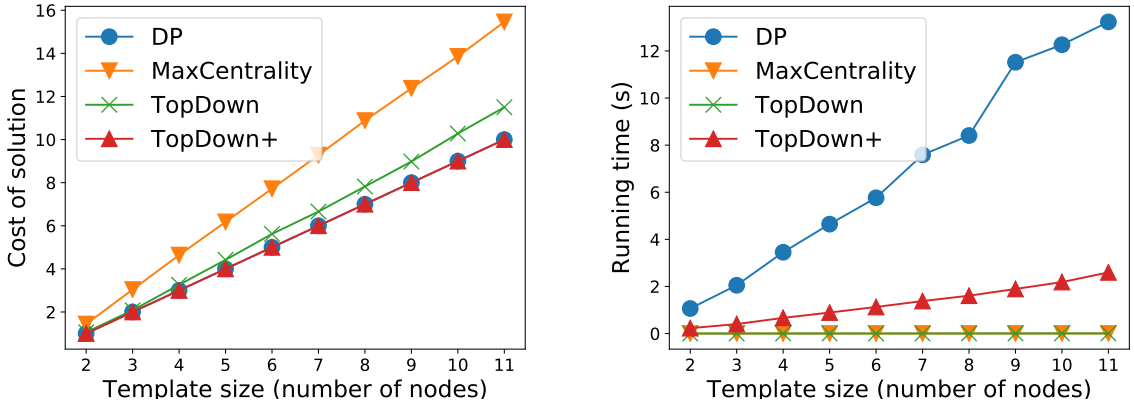
balance between *DP* and *MaxCentrality*. Its running time is slightly higher than that of *TopDown* but it is still reasonably low, while the solution cost is almost identical to that of the optimal *DP* algorithm.

5.3 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 will study the performance of the different heuristics, and also consider templates of larger size.

We consider two types of templates: The first is complete binary trees constructed in the same way as for TDTF-SUT, but of larger size; The second is full trees of height 2, with varying branching factor. The skill assignment is done in the same way as for the TDTF-SUT problem, ensuring that smaller templates are included in the larger ones, but now the same skill may appear in multiple positions. All reported results are averaged over 50 different experiments. For the *DPH* algorithm we report the averages for the experiments for which it produced a solution.

Figures 5.3, 5.4, and 5.5, 5.6 show the results of our experiments. We observe again that the *DPH* algorithm performs best in terms of solution cost but has also the highest running time. The running time of *DPH* scales linearly with respect to



(a) *Movies* solution cost

(b) *Movies* running time

Figure 5.2: Solution cost and running time for the TDTF-SUT problem with a CBT template for the *Movies* dataset.

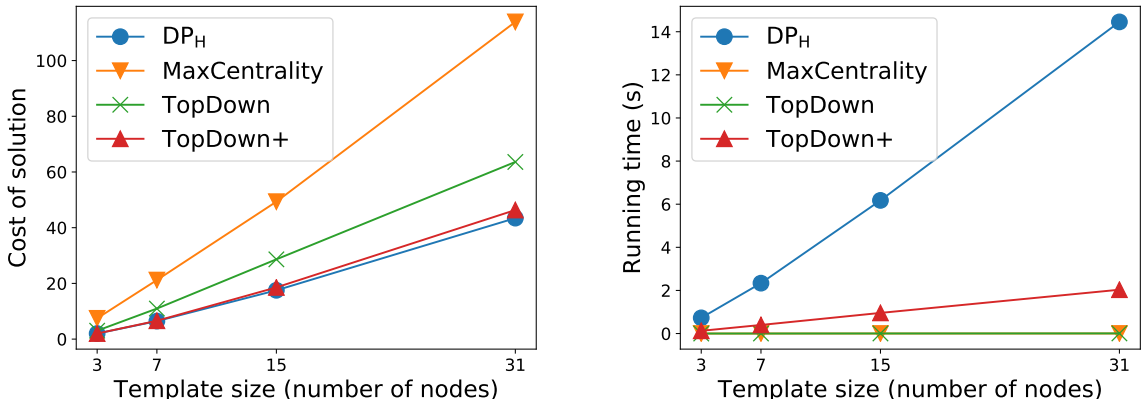
the size of the template, and quadratically with respect to the branching factor (given than the height of the tree is 2). The *TopDown+* algorithm is again the best option with low running time, and solution cost essentially identical to that of *DPH*.

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 we had failures for the *Academic* dataset, for large template size and large branching factor. More specifically for template size between 26 to 31 we had failures ranging from 2% to 16%, while for branching factor 4, we have 4% failures. These are non-negligible percentages, which demonstrate the weakness of *DPH* for large templates.

5.4 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 and other heuristics 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



(a) *Academic* solution cost

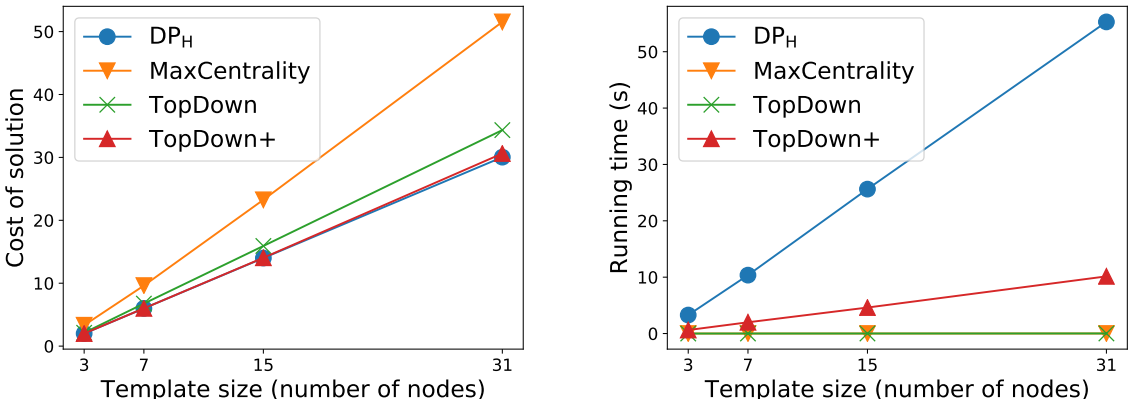
(b) *Academic* running time

Figure 5.3: Solution cost and running time for the TDTF-MRT problem with CBT template for varying template size for the *Academic* dataset.

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.

We considered the "flower" template for our experiments. Recall that 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. 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 5.7 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* is a good algorithm in practice for general templates.



(a) *Movies* solution cost

(b) *Movies* running time

Figure 5.4: Solution cost and running time for the TDTF-MRT problem with CBT template for varying template size for the *Movies* dataset.

5.5 Case Studies

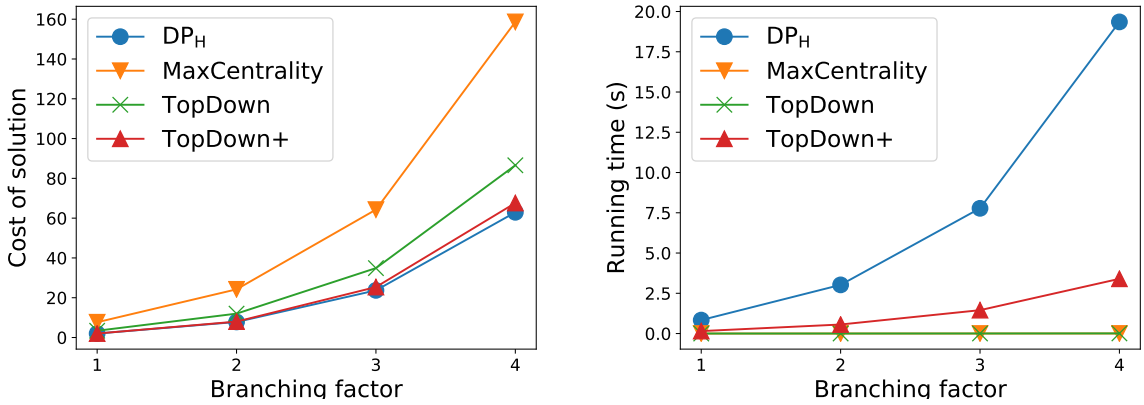
Finally, we perform two case studies, one for each dataset. We consider the TDTF-SUT problem, and we manually set the template skills and evaluate the results. Our goal is to empirically evaluate the solutions produced by the *DP* algorithm.

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 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*. Using the seniority attribute, we construct skills that use a combination of the seniority and a field of computer science.

The template we used in our experiment with *Academic* is shown in Figure 5.8a. 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. Each division head will manage two junior researchers in their respective field.

The result we obtain, shown in Figure 5.8b, is highly intuitive[3]. Ion Stoica, Pro-

[3]The seniority labels are debatable and also limited by the data provided by MS Academic. However,



(a) *Academic* solution cost

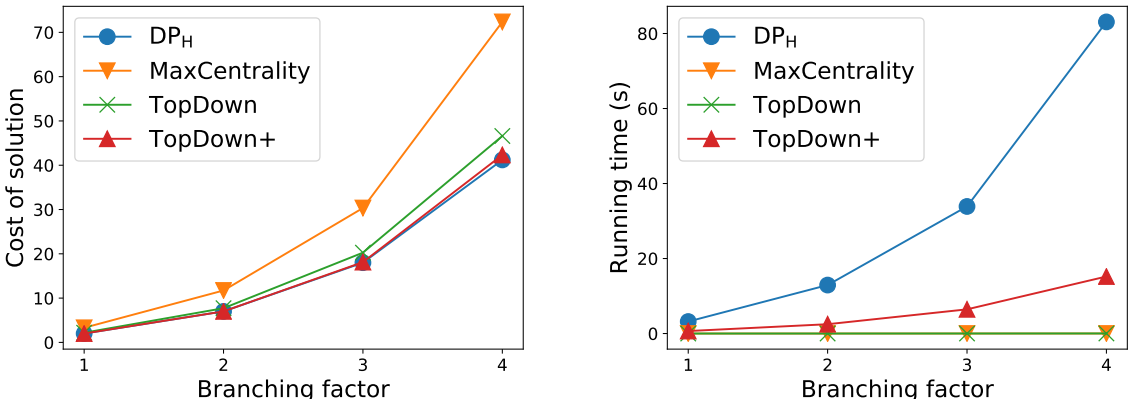
(b) *Academic* running time

Figure 5.5: Solution cost and running time for TDTF-MRT with a full tree template of height 2 and varying branching factor for the *Academic* dataset.

fessor at U.C. Berkeley, authority in the field of distributed systems with a broad set of interests, is the 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, long-time 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 5.9a. 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 5.9b. 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

the correct definition of seniority is beyond the scope of this thesis, and of the team formation problem.

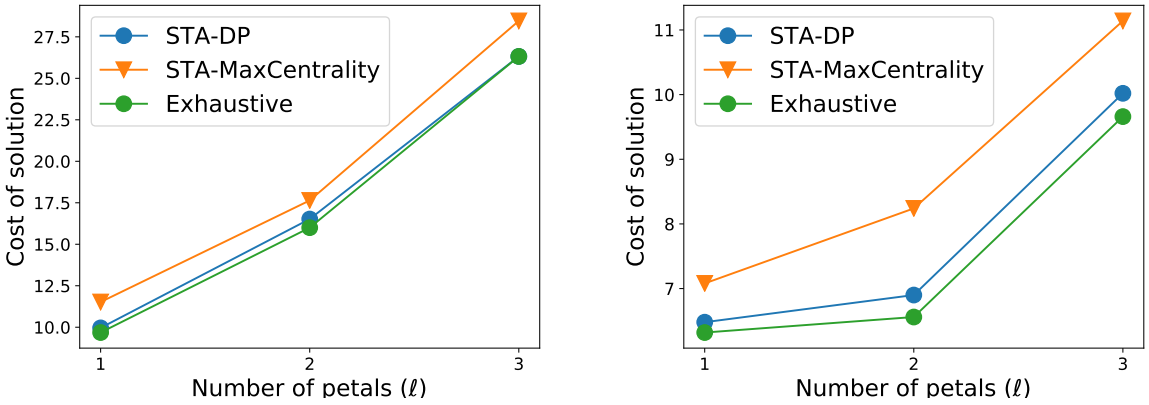


(a) *Movies* solution cost

(b) *Movies* running time

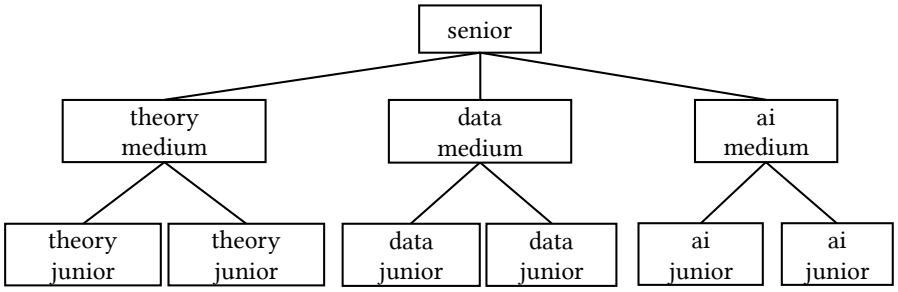
Figure 5.6: Solution cost and running time for TDTF-MRT with a full tree template of height 2 and varying branching factor for the *Movies* dataset.

intuitive.

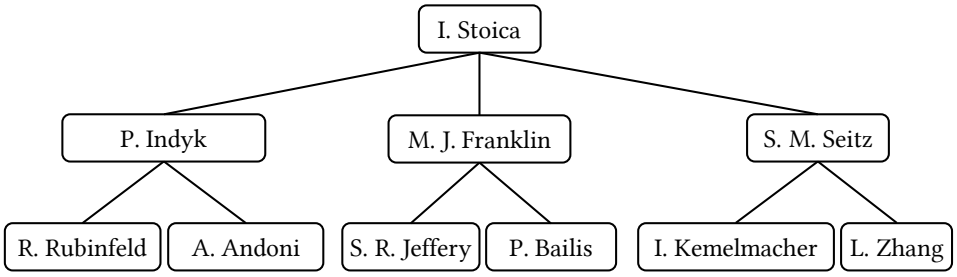


(a) *EgoKleinberg* (b) *EgoClooney*

Figure 5.7: Solution costs for flower graph templates, for varying number of petals (ℓ) with fixed petal size $k = 2$.

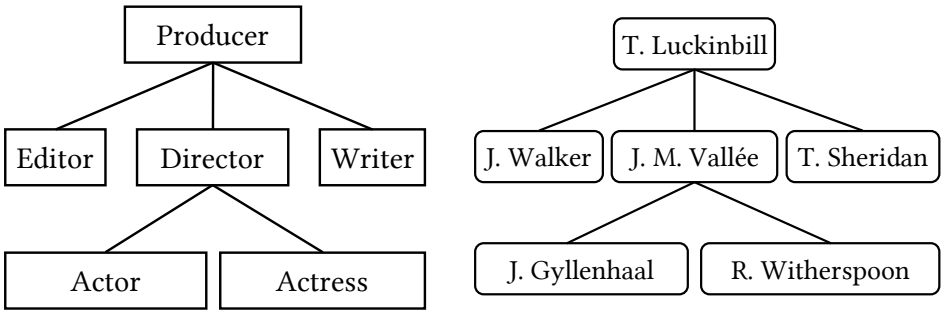


(a) Template & Skills



(b) *DP* Solution

Figure 5.8: *Academic* case study.



(a) Template & Skills

(b) *DP* Solution

Figure 5.9: *Movies* case study.

Chapter 6

Conclusion

In this thesis we introduced and studied the novel problem of template-driven team formation where, given a set of workers organized in an network, a set of skills for each worker, and a template team structure with required skills for each position, we seek to find a team of workers that have the required skills and minimize the communication cost over the edges of the template. 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 provide heuristic and approximate algorithms for the general case. Our experiments demonstrate that our algorithms are effective in practice. For future work, we are interested in better understanding how the properties of the template graph affect the team formation process, and derive approximation guarantees for more general families of graphs.

Bibliography

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

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

[3] R. Bredereck, JiehuaChen, F. Hüffner, and StefanKratsch, "Parameterized complexity of team formation in social networks," *Theoretical Computer Science*, vol. In press, 2017.

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

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

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

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

[8] 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.

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

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

[11] R. Agrawal, B. Golshan, and E. Terzi, "Grouping students in educational settings," in *ACM SIGKDD*, pp. 1017–1026, 2014.

[12] S. Bahargam, D. Erdös, A. Bestavros, and E. Terzi, "Personalized education; solving a group formation and scheduling problem for educational content," in *International Conference on Educational Data Mining, EDM*, pp. 488–491, 2015.

[13] M. Eftekhar, F. Ronaghi, and A. Saberi, "Team formation dynamics: A study using online learning data," in *ACM on Conference on Online Social Networks, COSN*, pp. 257–267, 2015.

[14] M. Wen, *Investigating Virtual Teams in Massive Open Online Courses: Deliberation-based Virtual Team Formation, Discussion Mining and Support*. PhD thesis, CMU, 2016.

[15] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu, "Graph pattern matching: From intractable to polynomial time," *Proc. VLDB Endow.*

[16] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2004.

[17] J. Li, Y. Cao, and S. Ma, "Relaxing graph pattern matching with explanations," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, 2017.

[18] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo, "Capturing topology in graph pattern matching," *CoRR*, 2012.

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

[20] M. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

Short Biography

Spiros Apostolou was born in Thessaloniki, Greece in 1990. He received his BSc degree from the Department of Computer Science & Engineering of University of Ioannina in 2015. During the same year, he became an MSc student at the same institution under the supervision of Panayiotis Tsaparas. Between 2015 and 2018, he took part in research programs, worked as a full stack developer for Terracom Informatics Ltd and worked on some personal projects of his. His research interests include data mining with a focus on graph and text mining and machine learning.