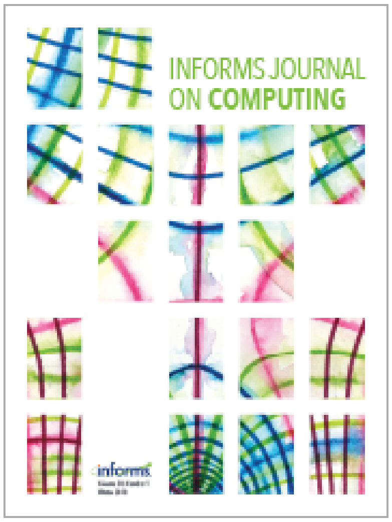


INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Branch-and-Bound Algorithm for Team Formation on Social Networks

Nihal Berktaş , Hande Yaman

informs
http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 33, No. 3, Summer 2021, pp. 1162–1176
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

A Branch-and-Bound Algorithm for Team Formation on Social Networks

Nihal Berktaş,[a] **Hande Yaman**[b]

[a] Department of Industrial Engineering, Bilkent University, 06800 Çankaya/Ankara, Turkey; [b] Research Centre for Operations Research and Statistics (ORSTAT), Faculty of Economics and Business, Katholieke Universiteit Leuven, Leuven 3000, Belgium

Contact: nihal.berktas@bilkent.edu.tr, https://orcid.org/0000-0002-3510-0808 (NB); hande.yaman@kuleuven.be, https://orcid.org/0000-0002-3392-1127 (HY)

Received: March 13, 2019
Revised: October 15, 2019; March 7, 2020; June 19, 2020
Accepted: June 25, 2020
Published Online in Articles in Advance: December 14, 2020

https://doi.org/10.1287/ijoc.2020.1000

Abstract. The team formation problem (TFP) aims to construct a capable team that can communicate and collaborate effectively. The cost of communication is quantified using the proximity of the potential members in a social network. We study a TFP with two measures for communication effectiveness; namely, we minimize the sum of communication costs, and we impose an upper bound on the largest communication cost. This problem can be formulated as a constrained quadratic set covering problem. Our experiments show that a general-purpose solver is capable of solving small and medium-sized instances to optimality. We propose a branch-and-bound algorithm to solve larger sizes: we reformulate the problem and relax it in such a way that it decomposes into a series of linear set covering problems, and we impose the relaxed constraints through branching. Our computational experiments show that the algorithm is capable of solving large-size instances, which are intractable for the solver.

Summary of Contribution: This paper presents an exact algorithm for the Team Formation Problem (TFP), in which the aim is, given a project and its required skills, to construct a capable team that can communicate and collaborate effectively. This combinatorial optimization problem is modeled as a quadratic set covering problem. The study provides a novel branch-and-bound algorithm where a reformulation of the problem is relaxed so that it decomposes into a series of linear set covering problems and the relaxed constraints are imposed through branching. The algorithm is able to solve instances that are intractable for commercial solvers. The study illustrates an efficient usage of algorithmic methods and modelling techniques for an operations research problem. It contributes to the field of computational optimization by proposing a new application as well as a new algorithm to solve a quadratic version of a classical combinatorial optimization problem.

History: Accepted by Andrea Lodi, Area Editor for Design and Analysis of Algorithms—Discrete.

Keywords: team formation problem • quadratic set covering • branch and bound • reformulation

1. Introduction

The complexity of products and services in today's world requires various skills, knowledge, and experience from different fields, whereas the pace of consumption demands agility in the production and development phases. To be able to meet these requirements, people are working in teams both physically and virtually in various organizations such as governments, nongovernmental organizations, universities, hospitals, and business firms. The quality of the work done depends on the technical capabilities of the team members and the effectiveness of communication among them. In the studies investigating the factors affecting the success of teams, communication has been considered to be one of the key factors, if not the most important one (Hoegl and Gemuenden 2001), especially in virtual teams (Jones 2005).

In addition to regular organizations that build physical and virtual teams for projects, there is a new concept of outsourcing called *team as a service*. The companies that use this model build a team according to the needs of a given project and provide managerial service throughout. The concept is claimed to provide the agility that companies need in today's fast-moving market because it reduces the burden on the core permanent employees by offering a self-sufficient team (Centric Digital 2016).

Motivated by this new concept of team as a service, we are interested in the team formation problem (TFP), which is the problem of selecting a group of people from a candidate set so that they work together on a given task that requires some technical skills. Our aim is to build a team whose members can collaborate effectively, and we do this by minimizing their communication cost.

In the operations research literature, the TFP has been studied in different contexts. The studies of Zakarian and Kusiak (1999) on product design, Boon and Sierksma (2003) on sports teams, and Agustín-Blas et al. (2011) on teaching groups are some examples in which the objective is to maximize the technical capability or the knowledge of the team. In the studies of Chen and Lin (2004), Fitzpatrick and Askin (2005), and Zhang and Zhang (2013), communication is taken into consideration using the personal characteristics of the team members. Well-known personality tests such as Myers-Briggs and Kolbe Conative are used to measure the effectiveness of communication. Baykasoglu et al. (2007) incorporate communication by specifying people who do not prefer to be in the same project. Gutiérrez et al. (2016) model interpersonal relations via the sociometric matrix, which consists of −1, 0, and 1 representing the negative, neutral, and positive relations, respectively. Another method to incorporate communication into the problem, the one chosen in this study, is via a social network of individuals. To the best of our knowledge, in the operations research literature, the study by Wi et al. (2009) is the first one to use social networks for team formation. The authors form a network using fuzzy familiarity scores among candidates via collaboration data and formulate a nonlinear program whose objective is a weighted sum of performance, familiarity, and size of the team. More recently, Farasat and Nikolaev (2016) use edge, two-star, three-star, and triangle network structures to measure the collaborative strength of the team. The objective is to maximize the weighted sum of structures in multiple teams, and the skills of people are not considered. The solution techniques suggested in these studies are either not designed for real-sized data or are heuristic approaches.

The TFPs where a social network is considered are mainly studied in the knowledge discovery and data-mining field, initiated by the work of Lappas et al. (2009) and followed by many others. This line of work is motivated by the existence of numerous online social networks and the advances in social network analysis. It uses a social network in which the edge weights are considered measures of the effort required for candidates to communicate as team members. Clearly, a lower weight for edge $\{i, j\}$ implies that candidates i and j can collaborate more effectively. Lappas et al. (2009) study two variants of the problem with different communication cost functions. The first is the diameter of the team, which is the largest distance between any pair of team members, where the distance between two people is taken as the shortest path weight in the network. The second function is the cost of a minimum-cost Steiner tree that spans the team members. Following this study, other functions are defined and used for the problem. The studies of Kargar and An (2011), Kargar et al. (2012), and Bhowmik et al. (2014) are among the ones that define the communication cost of the team as the sum of distances, which is the sum of the shortest path lengths between all pairs of team members. Kargar and An (2011) define leader distance as the sum of shortest path lengths between the leader and the person chosen for each required skill. Given a team, the bottleneck cost is defined by Majumder et al. (2012) as the maximum edge weight in a tree that minimizes this and that spans the team members. Dorn and Dustdar (2010) and Gajewar and Sarma (2012), by contrast, use communication cost functions that are related to the density of the team's subgraph.

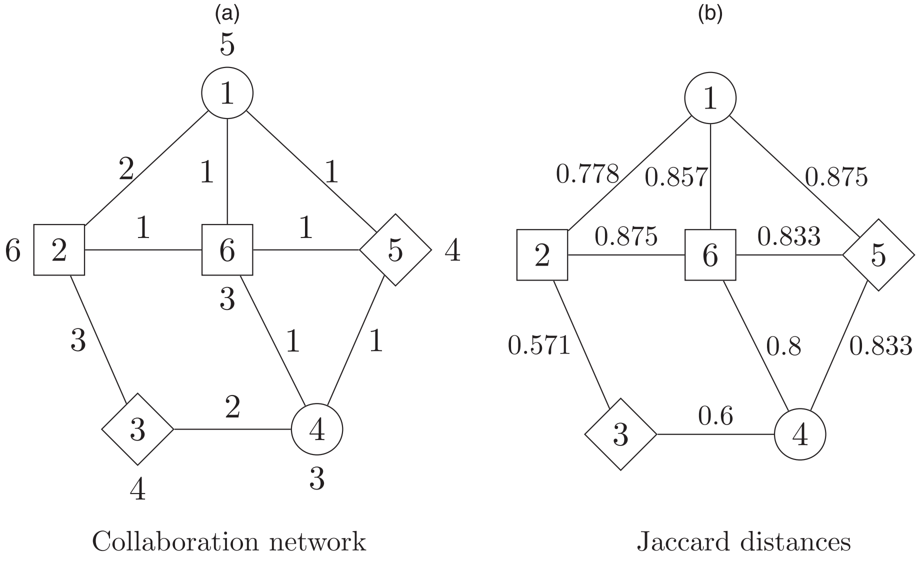
We adopt the problem definition of Lappas et al. (2009) and use a social network to quantify and minimize the communication cost. The technical capability of the team is ensured using a binary skill matrix built by considering minimum expertise levels. We propose to minimize the sum of distances and to impose an upper bound on the diameter. We derive a mixed-integer programming formulation for this new problem and test it using a large set of instances. We observe that small and medium-sized instances can be solved using a general-purpose solver, but memory problems occur for large instances. We present a novel branch-and-bound algorithm that is very effective in solving these instances.

The remaining part of the paper is organized as follows: In the next section, we formally define the TFP and provide quadratic and linear mathematical models. We present our branch-and-bound algorithm in Section 3. In Section 4, we first introduce our data sets and explain our instance-generation method. Then we present the results of an extensive computational study. We conclude in Section 5.

2. Problem Definition and Mathematical Models

In this section, we formally define the TFP, explain how the communication costs are computed, and provide mathematical models. Let K be the set of required skills for a given task, and let N be the set of candidates. We assume that the skills of the candidates are known. We need to select team members such that for each skill there is at least one person on the team who has that skill. Such teams are called *capable teams*. An undirected collaboration network of the candidates $G = (N, E)$ is given. In a collaboration network, two people (nodes) are connected by an edge if they have collaborated before. Edge $\{i, j\}$ has weight c_{ij}. These weights are commonly calculated in the following way: let i and j be two people and P_i and P_j be the sets of projects they have taken part in, respectively. Then $|P_i \cap P_j|$ is the number of their collaborations, and the weight of edge $\{i, j\}$ is taken as $1 - (|P_i \cap P_j|/|P_i \cup P_j|)$, which is the Jaccard metric, a well-known dissimilarity measure introduced by Jaccard (1912). The Jaccard distance between any two people with no collaboration equals one. Instead of taking the distance between all such unconnected pairs as one, Lappas et al. (2009) and others use the

Figure 1. Collaboration Network and Corresponding Jaccard Distances



Collaboration network

Jaccard distances

shortest path distances among these pairs. This method differentiates the unconnected pairs who have neighbors that collaborated often from the ones who have distant connections. We follow the same approach and define the cost of communication between i and j, denoted by p_{ij}, to be equal to c_{ij} if $P_i \cap P_j \neq \emptyset$, to be equal to the weight of the shortest path between i and j if $P_i \cap P_j = \emptyset$, and to be equal to a sufficiently large number if there is no path between them. By construction, all communication costs are nonnegative.

Before moving on to the problem definition, we demonstrate the cost-calculation procedure on a small example. In Figure 1, on the left, we have a collaboration network where the nodes represent people, and the shapes indicate the skill they have. The number next to each node is the total number of projects on which the person has worked. The number on each edge shows the number of collaborations of the people corresponding to the end nodes of the edge. The numbers on edges of the network on the right are the Jaccard distances calculated from the collaboration data. Then, calculating the shortest paths, we get the distance (communication cost) matrix in Table 1.

Table 1. Communication Cost Matrix for the People in the Collaboration Network

N	1	2	3	4	5	6
1	0	0.778	1.349	1.657	0.875	0.857
2	—	0	0.571	1.171	1.653	0.875
3	—	—	0	0.6	1.433	1.4
4	—	—	—	0	0.833	0.8
5	—	—	—	—	0	0.833
6	—	—	—	—	—	0

Under the setting given previously, the TFP is defined as finding a capable team with minimum communication cost. With communication costs computed as described, minimizing the sum of the distances amounts to maximizing the average familiarity of the team. There are empirical studies in the literature indicating positive effects of team familiarity on the performance of teams. The results of the study by Huckman et al. (2009) on a software service company indicate a positive and significant relation between team familiarity and operational performance. Analyzing software development teams of a telecommunications firm, Espinosa et al. (2007) find that team familiarity is more beneficial when coordination is more challenging because of team size or dispersion. The study by Avgerinos and Gokpinar (2016) on productivity of surgical teams also shows that the benefit of familiarity increases as the task gets more complex. Moreover, the performance analysis in the study suggests that the bottleneck pair, that is, the pair with the lowest familiarity, significantly reduces team productivity. In terms of the communication cost measures, the least familiar pair on a team amounts to the nodes whose distance equals the diameter of the team.

Motivated by the results of these studies, we choose to study the problem where we minimize the sum of distances and bound the diameter. We call this problem the *diameter-constrained TFP with sum-of-distances objective* (DC-TFP-SD).

In the remaining part of this section, we provide mathematical models for the DC-TFP-SD. For each person $i \in N$, we define a binary variable y_i to be one if this person is on the team and zero otherwise. We define parameter a_{ik} to be one if person $i \in N$ possesses skill $k \in K$ and to be zero otherwise. We let set C be the set of pairs of people in conflict, that is, the set of pairs whose communication cost exceeds the allowed diameter, and we eliminate teams that include such pairs. The DC-TFP-SD can be modeled as follows:

$$\min \sum_{i\in N}\sum_{j\in N: i<j} p_{ij} y_i y_j, \tag{1}$$

$$\text{subject to (s.t.)} \sum_{i\in N} a_{ik} y_i \geq 1, \quad \forall k \in K, \tag{2}$$

$$y_i + y_j \leq 1, \quad \forall \{i,j\} \in C, \tag{3}$$

$$y_i \in \{0,1\}, \quad \forall i \in N. \tag{4}$$

The covering Constraints (2) ensure that each required skill is covered; that is, there is at least one person on the team who has that skill. The family of packing (conflict) Constraints (3) forbids conflicting pairs on the team. The objective function is the sum of communication costs of team members.

We can use variables $z_{ij} = y_i y_j$ for all $i, j \in N$ with $i < j$ to linearize the objective function:

$$\min \sum_{i\in N}\sum_{j\in N: i<j} p_{ij} z_{ij}, \tag{5}$$

$$\text{s.t. } (2)\text{–}(4),$$

$$z_{ij} \geq y_i + y_j - 1, \quad \forall i, j \in N : i < j, \tag{6}$$

$$z_{ij} \leq y_i, \quad \forall i, j \in N : i < j, \tag{7}$$

$$z_{ij} \leq y_j, \quad \forall i, j \in N : i < j, \tag{8}$$

$$z_{ij} \geq 0, \quad \forall i, j \in N : i < j. \tag{9}$$

Constraints (6)–(9) are to linearize $z_{ij} = y_i y_j$ and force z_{ij} to be one when both y_i and y_j are equal to one and to be zero otherwise (Fortet 1960). Because the objective function coefficients are nonnegative, Constraints (7) and (8) can be dropped without changing the optimal value. One can use constraints $z_{ij} = 0$ for all $\{i,j\} \in C$ instead of Constraints (3), which give similar results in terms of computation time. Using both constraints together proved to be less effective.

If $C = \emptyset$, then we obtain the *team formation problem with sum-of-distances objective* (TFP-SD). The optimal solution of the TFP-SD on the network in Figure 1, with p_{ij} taken as in Table 1, is the team {2,3,4} with cost 2.342. The optimal solution of the DC-TFP-SD with a diameter limit of 0.9 is the team {4,5,6} with cost 2.466.

3. Branch-and-Bound Algorithms

The DC-TFP-SD is a quadratic set covering problem with side constraints (packing Constraints (3)). One of the earliest studies on the quadratic set covering problem is by Bazaraa and Goode (1975), where the authors propose a cutting-plane algorithm. Besides this study, the literature on quadratic set covering is limited to a study of polynomial approximations by Escoffier and Hammer (2007); a linearization technique by Saxena and Arora (1997), which does not guarantee optimality, as shown by Pandey and Punnen (2017); and a study by Punnen et al. (2019) on comparing different representations of the problem.

As listed in the surveys of Loiola et al. (2007) on the quadratic assignment problem and Pisinger et al. (2007) on the quadratic knapsack problem, the formulations of 0–1 quadratic problems can be based on mixed-integer, convex quadratic, or semidefinite programming, and mostly they are too large to be solved in their current forms. Therefore, they are relaxed and embedded into an algorithm such as a branch-and-bound, cutting-plane, or dual-ascent algorithm or a combination thereof. Most recent studies with semidefinite relaxations include the works of Povh and Rendl (2009), Mittelmann and Peng (2010), and de Klerk et al. (2015) on the quadratic assignment problem and the work of Guimarães et al. (2020) on the quadratic minimum spanning tree. Among the studies based on mixed-integer programming, see, for instance, a constraint-generation algorithm for the quadratic knapsack by Rodrigues et al. (2012), a branch-and-cut algorithm for the capacitated vehicle routing problem with quadratic objective by Martinelli and Contardo (2015), and a branch-and-price algorithm for the quadratic multiple knapsack by Bergman (2019).

As can be seen from this brief review, the quadratic set covering problem has attracted very little attention as opposed to other quadratic 0–1 problems. In this section, we first present a branch-and-bound algorithm for the TFP-SD, which is a quadratic set covering problem, and then extend it to the DC-TFP-SD, which is a quadratic set covering problem with side constraints.

3.1. Reformulation, Relaxation, and Decomposition

For ease of decomposition, we define variable z_{ij} for all $i, j \in N$ such that $i \neq j$ instead of $i < j$. We apply the idea of the well-known reformulation-linearization technique (RLT) of Adams and Sherali (1986) to derive the following inequalities from the original covering constraints by multiplying each one by variable y_j:

$$\sum_{i\in N\setminus\{j\}} a_{ik} z_{ij} \geq (1 - a_{jk}) y_j, \quad \forall k \in K, j \in N.$$

The right-hand side of this constraint is equal to one when person j is on the team but does not have skill k. Hence, the constraint implies that in this case, at least one person having skill k must be on the team. We can rewrite these constraints as follows:

$$\sum_{i\in N\setminus\{j\}} a_{ik} z_{ij} \geq y_j, \quad \forall k \in K, j \in N : a_{jk} = 0. \tag{10}$$

We call these new constraints RLT constraints. By adding these into our previous model and making slight changes, we obtain the following reformulation of the TFP-SD:

$$\min \frac{1}{2}\sum_{i\in N}\sum_{j\in N\setminus\{i\}} p_{ij}z_{ij}$$

$$\text{s.t. } (2),\ (4),\ (10),$$

$$z_{ij} \le y_j, \quad \forall i,j \in N : i \ne j, \tag{11}$$

$$z_{ij} = z_{ji}, \quad \forall i,j \in N : i < j, \tag{12}$$

$$z_{ij} \ge y_i + y_j - 1, \ \forall i,j \in N : i < j, \tag{13}$$

$$z_{ij} \in \{0,1\}, \quad \forall i,j \in N : i \ne j. \tag{14}$$

In the reformulation, we use constraints $z_{ij} \in \{0,1\}$ rather than $z_{ij} \ge 0$ for all $i,j \in N$ with $i \ne j$ even though the latter constraints are also sufficient to have a correct formulation. However, in what follows, we will relax some constraints, and the integrality of z-variables will not be implied in the relaxed problem.

There are many studies on using RLT to solve quadratic problems. In the works of Adams et al. (2007) and Hahn et al. (2012), different levels of RLT are used for the quadratic assignment problem. In these studies, Lagrangian relaxation is applied to the reformulations and embedded into a branch-and-bound algorithm. The technique is also used for the quadratic knapsack problem by Billionnet and Calmels (1996), Caprara et al. (1999), Pisinger et al. (2007), and Fomeni et al. (2014). The main distinction between these reformulations and ours is that constraints of type (13) are redundant in these reformulations because of problem and cost structure, whereas in our case they are necessary.

We are interested in the relaxation of the reformulation obtained by removing Constraints (12) and (13). Let (y^*, z^*) be an optimal solution of the relaxation. Because Constraints (12) are relaxed, z^*_{ij} may not be equal to z^*_{ji}. Furthermore, we might get a solution where $z^*_{ij} \ne y^*_i y^*_j$ or $z^*_{ji} \ne y^*_i y^*_j$ or both because we relaxed Constraints (13). To remove such infeasibilities, we branch by creating two nodes: at one node, we allow at most one of i and j to be on the team, and at the other node, we force both to be on the team. Suppose now that we are at node ℓ of the branch-and-bound tree, and thus far, while branching, we have added the constraints that at most one of i and j can be on the team for all $\{i,j\} \in C^1_\ell$ (by adding the constraints $y_i + y_j \le 1$, $z_{in} + z_{jn} \le y_n$ for all $n \in N \setminus \{i,j\}$ and $z_{ij} = z_{ji} = 0$) and that i and j are both on the team for all $\{i,j\} \in C^2_\ell$ (by adding the constraints $y_i = y_j = 1$, $z_{in} = z_{jn} = y_n$ for all $n \in N \setminus \{i,j\}$ and $z_{ij} = z_{ji} = 1$). Then the relaxation at node ℓ, called R_ℓ, is as follows:

$$\min \frac{1}{2}\sum_{i\in N}\sum_{j\in N\setminus\{i\}} p_{ij}z_{ij}$$

$$\text{s.t. } (2),\ (4),\ (10),\ (11),\ (14),$$

$$y_i + y_j \le 1, \quad \forall\{i,j\} \in C^1_\ell, \tag{15}$$

$$y_i = y_j = 1, \quad \forall\{i,j\} \in C^2_\ell, \tag{16}$$

$$z_{in} + z_{jn} \le y_n, \quad \forall\{i,j\} \in C^1_\ell, n \in N \setminus \{i,j\}, \tag{17}$$

$$z_{in} = z_{jn} = y_n, \quad \forall\{i,j\} \in C^2_\ell, n \in N \setminus \{i,j\}, \tag{18}$$

$$z_{ij} = z_{ji} = 0, \quad \forall\{i,j\} \in C^1_\ell, \tag{19}$$

$$z_{ij} = z_{ji} = 1, \quad \forall\{i,j\} \in C^2_\ell. \tag{20}$$

Next we show that R_ℓ can be solved by solving $|N| + 1$ linear set covering problems with side constraints (see, e.g., Caprara et al. 1999 for a similar result for the quadratic knapsack problem).

Proposition 1. *The relaxation R_ℓ can be solved by solving $|N| + 1$ linear set covering problems with side constraints as follows. For each $n \in N$, we solve the linear set covering problem (Pr_n), which will be referred to as subproblem n:*

$$v_n = \min \sum_{i\in N\setminus\{n\}} p_{in}\zeta^n_i \tag{21}$$

$$\text{s.t.} \sum_{i\in N\setminus\{n\}} a_{ik}\zeta^n_i \ge 1, \quad \forall k \in K : a_{nk} = 0,$$

$$\zeta^n_i + \zeta^n_j \le 1, \quad \forall\{i,j\} \in C^1_\ell : i,j \ne n, \tag{22}$$

$$\zeta^n_i = \zeta^n_j = 1, \quad \forall\{i,j\} \in C^2_\ell : i,j \ne n, \tag{23}$$

$$\zeta^n_i = 0, \quad \forall\{i,n\} \in C^1_\ell, \tag{24}$$

$$\zeta^n_i = 1, \quad \forall\{i,n\} \in C^2_\ell, \tag{25}$$

$$\zeta^n_i \in \{0,1\}, \quad \forall i \in N \setminus \{n\} \tag{26}$$

with optimal solution $\bar\zeta^n$ and optimal value v_n. Then the optimal value of R_ℓ can be computed by solving the following master problem:

$$v = \min \frac{1}{2}\sum_{j\in N} v_j y_j$$

$$\text{s.t.} \sum_{j\in N} a_{jk}y_j \ge 1, \quad \forall k \in K,$$

$$y_i + y_j \le 1, \quad \forall\{i,j\} \in C^1_\ell,$$

$$y_i = y_j = 1, \quad \forall\{i,j\} \in C^2_\ell,$$

$$y_j \in \{0,1\}, \quad \forall j \in N.$$

Moreover the solution (y^*, z^*), where y^* is an optimal solution of the master problem and $z^*_{ij} = y^*_j \bar\zeta^j_i$ for all $i,j \in N : i \ne j$, is an optimal solution for R_ℓ.

Proof. It is sufficient to observe that in R_ℓ, for a given vector $\boldsymbol{y}$, the problem of computing the best $\boldsymbol{z}$ decomposes into subproblems, one for each $n \in N$ with $y_n = 1$. When $y_n = 1$, the best values of z_{in} are $z_{in} = \bar\zeta^n_i$ for all $i \in N \setminus \{n\}$. Then the best $\boldsymbol{y}$ can be computed by solving the preceding master problem. □

We note that we can also multiply Constraints (2) with $(1 - y_j)$ for $j \in N$ and obtain valid inequalities $\sum_{i\in N\setminus\{j\}} a_{ik}(y_i - z_{ij}) \ge 1 - y_j$ for $k \in K$ after substituting $z_{ij} = y_i y_j$ for $i \in N \setminus \{j\}$ and $y_j(1 - y_j) = 0$. However, if we add these constraints to our reformulation, then the relaxed problem does not decompose any more.

In our branch-and-bound algorithm, we propose to work with a weaker relaxation R'_ℓ, which is obtained by dropping Constraints (17) and (18) in R_ℓ. The relaxation R'_ℓ can be solved by solving for each $n \in N$ the *relaxed subproblem* Pr'_n, which is obtained from subproblem Pr_n by dropping Constraints (22) and (23), with optimal solution $\bar{\zeta}'^n$ and optimal value v'_n, and then by solving the *relaxed master problem*, whose optimal value is ν' and in which v_j is replaced by v'_j in the objective function.

At the root node $\ell = 0$, R'_0 is the same as R_0 and is solved by solving $|N| + 1$ linear set covering problems. We need less computation at the other nodes, as we explain next in Proposition 2.

Proposition 2. *At node ℓ of the branch-and-bound tree where ℓ is not the root node, the relaxation R'_ℓ can be solved by solving at most three linear set covering problems with side constraints if the optimal solutions and optimal values of the subproblems at the parent node are available.*

Proof. Let ℓ' be the parent node of node ℓ. Suppose that the we obtained the current node by adding $\{i', j'\}$ to $C^1_{\ell'}$, that is, $C^1_\ell = C^1_{\ell'} \cup \{i', j'\}$ and $C^2_\ell = C^2_{\ell'}$. Then we add the constraint $y_{i'} + y_{j'} \le 1$ to the master problem $\zeta^{j'}_{i'} = 0$ to the relaxed subproblem $\text{Pr}'_{j'}$, $\zeta^{i'}_{j'} = 0$ to the relaxed subproblem $\text{Pr}'_{i'}$, and the other subproblems remain unchanged. If the optimal solution of $\text{Pr}'_{i'}$ (respectively, $\text{Pr}'_{j'}$) at node ℓ' satisfies $\zeta^{i'}_{j'} = 0$ (respectively, $\zeta^{j'}_{i'} = 0$), then it is also optimal for subproblem $\text{Pr}'_{i'}$ (respectively, $\text{Pr}'_{j'}$) at node ℓ. Otherwise, we solve these subproblems and then we solve the master problem with the additional constraint $y_{i'} + y_{j'} \le 1$. If the current node is obtained by adding $\{i', j'\}$ to C^2_ℓ, then again we may need to solve the relaxed subproblems $\text{Pr}'_{i'}$ and $\text{Pr}'_{j'}$ with the additional constraints $\zeta^{i'}_{j'} = 1$ and $\zeta^{j'}_{i'} = 1$, respectively, and then the master problem with $y_{i'} = 1$ and $y_{j'} = 1$. □

As in R_ℓ, the solution $(\boldsymbol{y}^*, \boldsymbol{z}^*)$, where $\boldsymbol{y}^*$ is an optimal solution of the relaxed master problem and $z^*_{ij} = y^*_j \bar{\zeta}'^j_i$ for all $i, j \in N : i \ne j$, where $\bar{\zeta}'^j$ is an optimal solution of the relaxed subproblem $\text{Pr}'_{j'}$, is an optimal solution for R'_ℓ.

The lower bound we get from R'_ℓ may not be as good as the lower bound of R_ℓ, and consequently, the branch-and-bound tree may be larger. However, our preliminary analysis has shown that this approach is faster because the time spent at each node is significantly smaller.

3.2. Branching Strategy

We should be able to eliminate a solution of the relaxation if it is not feasible for the original problem. We do this by branching. In Observation 1, we present different cases of infeasibility.

Observation 1. If the optimal solution $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ to the relaxation R'_ℓ at node ℓ is not feasible for the original problem at node ℓ, then there exists at least one pair $\{i, j\}$ satisfying one of the following conditions:

- $y^*_i = y^*_j = 1$ and $z^*_{ij} = z^*_{ji} = 0$ (type 1 pair), or
- $y^*_i = y^*_j = 1$, $z^*_{ij} = 1$, and $z^*_{ji} = 0$ (type 2 pair), or
- $y^*_i = 1$, $y^*_j = 0$, $z^*_{ij} = 0$, and $z^*_{ji} = 1$.

We only branch on type 1 or type 2 pairs by prioritizing the former. If the current solution is not feasible, we branch on the first type 1 pair we find. If none exists, we branch on the first type 2 pair (see Algorithm 1). Next, in Proposition 3, we show that branching on only type 1 and type 2 pairs is sufficient.

Proposition 3. *If the optimal solution $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ to the relaxation R'_ℓ at node ℓ is not feasible for the original problem at node ℓ, then there exists either a type 1 pair or a type 2 pair or $(\boldsymbol{y}^*, \bar{z})$ where $\bar{z}_{ij} = y^*_i y^*_j$ for all $i, j \in N$ such that $i \ne j$ is an alternate optimal solution to the relaxation R'_ℓ.*

Proof. Suppose that there is no type 1 or type 2 pair in $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ and the solution $(\boldsymbol{y}^*, \bar{z})$ is not an alternate optimal solution to the relaxation R'_ℓ. Then, by Observation 1, there exists at least one pair $\{i, j\}$ such that $y^*_i = 1$, $y^*_j = 0$, $z^*_{ij} = 0$, and $z^*_{ji} = 1$. Because $(\boldsymbol{y}^*, \bar{z})$ is not an alternate optimal solution, for one of such pairs, setting z_{ji} to zero violates a constraint. Then there exists a skill k that is covered uniquely by j in the relaxed subproblem Pr'_i because otherwise setting z_{ji} to zero would be feasible. Because $y^*_j = 0$, skill k is covered by another candidate, for example, candidate t, in the relaxed master problem. Therefore, $y^*_t = 1$. However, $\bar{\zeta}'^i_t$ and consequently z^*_{ti} must be zero because k is covered uniquely by j in subproblem Pr'_i. Then $\{i, t\}$ is a pair with $y^*_i = y^*_t = 1$ and $z^*_{ti} = 0$ and is either a type 1 or type 2 pair. This contradicts our assumption. □

Algorithm 1 (BranchPair($\boldsymbol{y}^*$; $\boldsymbol{z}^*$))

```
for $i \in N : y^*_i = 1$, do
    for $j \in N : j > i, y^*_j = 1$, do
        if $z^*_{ij} = z^*_{ji} = 0$, then
            pair ← $\{i, j\}$;
            break
if pair = null, then
    for $i \in N : y^*_i = 1$, do
        for $j \in N : j > i, y^*_j = 1$, do
            if $z^*_{ij} \ne z^*_{ji}$, then
                pair ← $\{i, j\}$;
                break
Return pair
```

3.3. Upper Bounds

There are two ways to update the upper bound in our algorithm: via the subproblems and via the master problem.

Proposition 4. *Let $N_j = \{i \in N : \bar{\zeta}'^j_i = 1\} \cup \{j\}$, where $\bar{\zeta}'^j$ is an optimal solution to the relaxed subproblem Pr'_j for $j \in N$,* *and $N' = \{i \in N : y_i^* = 1\}$, where $\boldsymbol{y}^*$ is an optimal solution of the relaxed master problem solved at any node of the branch-and-bound tree. Then $u^j = 1/2 \sum_{i' \in N_j} \sum_{j' \in N_j \setminus \{i'\}} p_{i'j'}$ for $j \in N$ and $u^0 = 1/2 \sum_{i' \in N'} \sum_{j' \in N' \setminus \{i'\}} p_{i'j'}$ are upper bounds for the optimal value.*

Proof. For each $j \in N$, because of Constraints (10) in the relaxed subproblem, N_j is a capable team. Similarly, because of Constraints (2) in the master problem, N' is also a capable team. Their sum of distance values give upper bounds. □

At each node, after solving the relaxed subproblems and the master problem, we update the upper bound and the incumbent solution if we find a better solution.

3.4. The Algorithm

The branch-and bound-algorithm is presented in Algorithm 2. The current lower and upper bounds are denoted as LB and UB. At each node ℓ, we keep the optimal solution of the subproblem $\ell.\bar{\zeta}'^n$ of Pr'_n, its optimal value $\ell.v'_n$ for all $n \in N$, the optimal value of the relaxed master problem $\ell.\nu'$, and its optimal solution $(\ell.\boldsymbol{y}^*, \ell.\boldsymbol{z}^*)$.

The initial step is to create the root node, 0, at which we solve the relaxed subproblems Pr'_n for all $n \in N$, and then the relaxed master problem, whose optimal value becomes the first lower bound. Because we preprocess our instances, we do not need to check for feasibility at the root node. As explained in Proposition 4, each time a relaxed subproblem or a relaxed master problem is solved, we check whether we can update the upper bound and the incumbent solution, team T. If $LB < UB$, then we initialize the queue Q by adding the root node.

The algorithm runs until the lower bound is equal to the upper bound. We follow the best-first search rule for choosing the next node to process, breaking ties arbitrarily. Let ℓ be a node in Q with the lowest lower bound. We remove ℓ from the queue and find its branch pair, say $\{i, j\}$. We create child nodes ℓ_1 and ℓ_2 and solve relaxations R'_{ℓ_1} and R'_{ℓ_2}, as explained in Proposition 2. Node ℓ_1 (respectively, ℓ_2) is added to the queue only if $\ell_2.\nu'$ (respectively, $\ell_1.\nu'$) is less than the current upper bound.

Throughout the algorithm, when a relaxed subproblem or a relaxed master problem is infeasible, its objective value is set to infinity. Therefore, if R'_ℓ is infeasible, then $\ell.\nu' = \infty$. In this case, we discard node ℓ because it does not satisfy $\ell.\nu' < UB$. This amounts to pruning by infeasibility. Furthermore, if the solution $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ of relaxation R'_ℓ is feasible for the original problem or is not feasible but $(\boldsymbol{y}^*, \bar{z})$ where $\bar{z}_{ij} = y_i^* y_j^*$ for all $i, j \in N$ such that $i \neq j$ is an alternate optimal solution to R'_ℓ, then $\ell.\nu' \geq UB$ because these solutions are used to update the upper bound. This corresponds to pruning by optimality. If the node is not pruned by infeasibility or optimality and $\ell.\nu' \geq UB$, then the node is pruned by bound. Hence, if a node is added to the queue, then it satisfies $\ell.\nu' < UB$ and has at least one type 1 or type 2 branch pair.

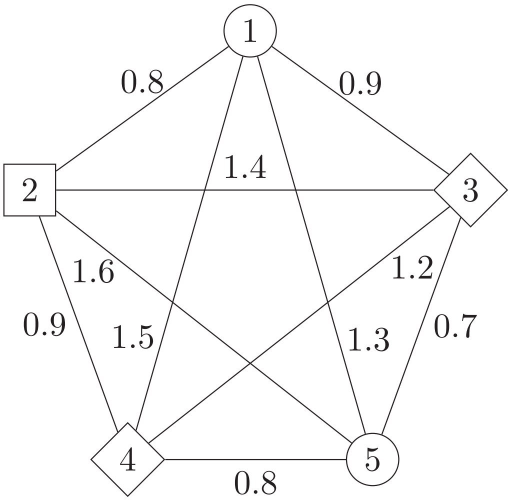
Algorithm 2 (Branch and Bound)

```
$UB := \infty$, $T = \emptyset$.
Create root node 0 with $0.\nu' := \infty, C_0^1 := \emptyset, C_0^2 := \emptyset$.
for $n \in N$, do
    Solve $\mathrm{Pr}'_n$.
    $0.\bar{\zeta}'^n := \bar{\zeta}'^n$ and $0.v'_n := v'_n$ ▷ update $UB$ and $T$
        if possible.
Solve the relaxed master problem.
$0.\boldsymbol{y}^* := \boldsymbol{y}^*, 0.\boldsymbol{z}^* := \boldsymbol{z}^*, 0.\nu' := \nu', LB := \nu'$ ▷ update
    $UB$ and $T$ if possible
if $LB < UB$, then $Q := \{0\}$
while $LB < UB$, do
    $\ell = \mathrm{argmin}_{\ell' \in Q}\{\ell'.\nu'\}, Q := Q \setminus \{\ell\}$
    $\{i, j\} := BranchPair(\ell.\boldsymbol{y}^*, \ell.\boldsymbol{z}^*)$.
    Create node $\ell_1$: $\ell_1.v'_n = \ell.v'_n, \ell_1.\bar{\zeta}'^n = \ell.\bar{\zeta}'^n, \forall n \in N,$
        $\ell_1.\nu' := \infty, C^1_{\ell_1} := C^1_\ell \cup \{i, j\}, C^2_{\ell_1} := C^2_\ell$.
    if $\ell.\bar{\zeta}'^i_j = 1$, then
        Solve $\mathrm{Pr}'_i$.
        if feasible, then $\ell_1.v'_i := v'_i, \ell_1.\bar{\zeta}'^i := \bar{\zeta}'^i$, else
            $\ell_1.v'_i := \infty$ ▷ update $UB$ and $T$ if possible.
    if $\ell.\bar{\zeta}'^j_i = 1$, then
        Solve $\mathrm{Pr}'_j$.
        if feasible, then $\ell_1.v'_j := v'_j, \ell_1.\bar{\zeta}'^j := \bar{\zeta}'^j$, else
            $\ell_1.v'_j := \infty$ ▷ update $UB$ and $T$ if possible
    Solve relaxed master problem
    if feasible, then $\ell_1.\boldsymbol{y}^* := \boldsymbol{y}^*, \ell_1 \boldsymbol{z}^* := \boldsymbol{z}^*,$
        $\ell_1.\nu' = \nu'$ ▷ update $UB$ and $T$ if possible.
    if $\ell_1.\nu' < UB$, then $Q := Q \cup \{\ell_1\}$.
    Create node $l_2$: $\ell_2.v'_n = \ell.v'_n$.
        $\ell_2.\bar{\zeta}'^n = \ell.\bar{\zeta}'^n, \forall n \in N,$
        $\ell_2.\nu' = \infty, C^1_{\ell_2} := C^1_\ell \, C^2_{\ell_2} := C^2_\ell \cup \{i, j\}$.
    if $\ell.\bar{\zeta}'^i_j = 0$, then
        Solve $\mathrm{Pr}'_i$.
        if feasible, then $\ell_2.v'_i := v'_i, \ell_2.\bar{\zeta}'^i := \bar{\zeta}'^i$, else
            $\ell_2.v'_i := \infty$ ▷ update $UB$ and $T$ if possible
    if $\ell.\bar{\zeta}'^j_i = 0$, then
        Solve $\mathrm{Pr}'_j$.
        if feasible, then $\ell_2.v'_j := v'_i, \ell_2.\bar{\zeta}'^j := \bar{\zeta}'^j$, else
            $\ell_2.v'_j := \infty$ ▷ update $UB$ and $T$ if possible.
    Solve relaxed master problem.
    if feasible, then $\ell_2.\boldsymbol{y}^* := \boldsymbol{y}^*, \ell_2.\boldsymbol{z}^* := \boldsymbol{z}^*,$
        $\ell_2.\nu' := \nu'$ ▷ update $UB$ and $T$ if possible.
    if $\ell_2.\nu' < UB$, then $Q := Q \cup \{\ell_2\}$.
    $LB := \min_{\ell' \in Q}\{\ell'.\nu'\}$.
Return $UB$ and $T$.
```

3.5. Example

We illustrate the branch-and-bound algorithm on a small example. We would like to solve the TFP-SD on the social network given in Figure 2. There are five

Figure 2. Example Network, Optimal Solutions of the Subproblems and the Master and the Bounds at the Root Node



problem	team	cost
Pr'_1	{1,2,3}	3.1
Pr'_2	{1,2,4}	3.2
Pr'_3	{2,3,5}	3.7
Pr'_4	{2,4,5}	3.3
Pr'_5	{2,3,5}	3.7
master	{1,2,4}	3.2
lb=2.55 ub=3.1		

candidates, and the shortest path lengths are as shown on the edges. The project requires three skills, and the skills of people are indicated by the shape of nodes.

At the root node of the branch-and-bound tree, we solve relaxation $R_0 = R'_0$, which requires solving five subproblems and then a master problem. In Figure 2, we summarize the information we get from these problems in the table next to the network. For example, the first row shows that the optimal solution of subproblem 1 is $\bar{\zeta}^1_2 = \bar{\zeta}^1_3 = 1$. The team consisting of persons 1, 2, and 3 has a cost 3.1. This is the upper bound we get from this subproblem, and actually, it is the best bound among all subproblems, so the corresponding solution becomes the incumbent. The solution of the master problem is $y^*_1 = y^*_2 = y^*_4 = 1$ and $y^*_3 = y^*_5 = 0$ with objective value of 2.55. This becomes the lower bound. We check whether we can use the solution of the master problem to update the upper bound. The team {1,2,4} costs 3.2, which is greater than the upper bound we get from subproblem 1, so the incumbent stays as {1,2,3}.

The entire branch-and-bound tree is illustrated in Figure 3. Next to each node, we summarize the solution and bound information in a table, similar to the one in Figure 2.

The solution at the root node is optimal unless we find a branch pair. Among i and j with $y^*_i = y^*_j = 1$, we first look for a pair with $z^*_{ij} = z^*_{ji} = 0$. Then {1,4} becomes our first branch pair. At the odd-numbered nodes, we ensure that the people in the branch pair are not teammates, and at the even-numbered nodes, they are forced to be on the team together. Therefore, at node 1, the problem R'_1 has the sets $C^1_1 = \{\{1,4\}\}$ and $C^2_1 = \emptyset$. At node 2, problem R'_2 has $C^1_2 = \emptyset$ and $C^2_2 = \{\{1,4\}\}$.

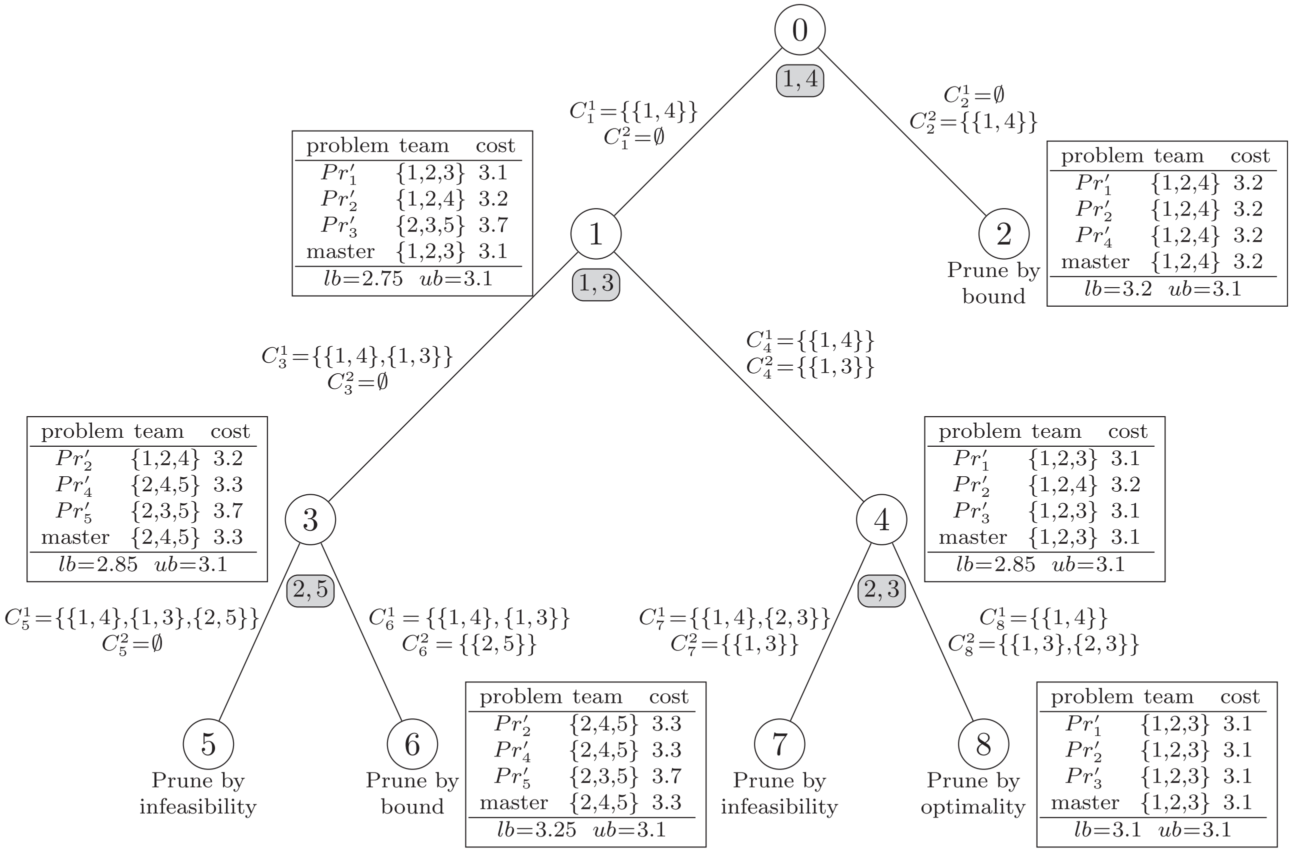
At node 1, we only solve the relaxed master problem because the solution of the relaxed subproblem 1 (respectively, 4) already satisfies $\bar{\zeta}'^1_4 = 0$ (respectively, $\bar{\zeta}'^4_1 = 0$). The optimal solution of the relaxed master problem is team {1,2,3}, and the lower bound we get at this node is 2.75. We do not update the upper bound because no better solution has been found. At node 2, we solve both relaxed subproblems, update v'_1 and v'_4, and solve the relaxed master problem. Because the lower bound we get at this node is greater than the current incumbent, we prune the node by bound. The algorithm continues with node 1, and the next branch pair becomes {1,3}, which is a type 2 pair. We create node 3 and problem R'_3 with $C^1_3 = \{\{1,4\},\{1,3\}\}$ and $C^2_3 = \emptyset$. We solve the relaxed subproblem 1 at this node, update v'_1, and solve the relaxed master problem. The lower bound at this node becomes 2.85. At node 4, we create problem R'_4 with $C^1_4 = \{\{1,4\}.$ and $C^2_4 = \{\{1,3\}\}$. We solve the relaxed subproblem 3, update v'_3, and then solve the relaxed master problem, which gives the same lower bound as node 3. We can continue with either of them, so we choose node 3, and the branch pair is {2,5}. At node 5, we create problem R'_5 with $C^1_5 = \{\{1,4\},\{1,3\},\{2,5\}\}$ and $C^2_5 = \emptyset$. We solve relaxed subproblem 5 and update v'_5, but the relaxed master problem becomes infeasible, and we prune the node. Continuing in this manner, the algorithm terminates at node 8, proving that the upper bound 3.1 found at the root node is actually the optimal value.

3.6. Branch-and-Bound Algorithm for the DC-TFP-SD

We can use a similar branch-and-bound algorithm to solve the DC-TFP-SD by making two adjustments. The first adjustment is in the relaxation that we solve to compute a lower bound, and the second adjustment is in the way we update upper bounds.

Recall that C is the set of pairs in conflict, and we forbid them by Constraints (3) in the formulation of

Figure 3. Branch-and-Bound Tree



the DC-TFP-SD. Also recall that R'_ℓ is the weaker relaxation of the reformulation of the TFP-SD at node ℓ of the branch-and-bound tree.

For the DC-TFP-SD, we can treat the conflict Constraints (3) like the constraints we use in branching and add them to the master and related subproblems. However, our preliminary analysis has shown that it is better to work with a further relaxation. We define R''_ℓ to be the relaxation obtained by adding Constraints (19) for all $\{i,j\} \in C$ to R'_ℓ. In other words, we add the conflict constraint for pair $\{i,j\} \in C$ to the subproblems i and j and not to the other subproblems nor the master. As a result, we have weaker lower bounds, but we work with a smaller master problem.

The second adjustment is in the upper bounding procedure. In Proposition 4, we define the set N_j for $j \in N$ and N' by the solutions of subproblem j and the master problem, respectively. For the TFP-SD, the teams defined by these sets were capable teams, so their cost values u^j for $j \in N$ and u^0 gave upper bounds. In the DC-TFP-SD, these are still capable teams, but they might have a pair in conflict. Thus, the second adjustment in the algorithm is to check the feasibility of these teams. If these teams have no pairs in conflict, their cost values are upper bounds for the optimal value of the DC-TFP-SD.

Using the relaxation R''_ℓ and this upper bounding procedure, we obtain valid lower and upper bounds. Next, we prove that if the optimal solution $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ that we obtain by solving R''_ℓ does not satisfy the conflict Constraints (3), then there exists a type 1 pair on which we can branch.

Proposition 5. *Let $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ be the optimal solution of R''_ℓ. If there exists a pair $\{i,j\} \in C$ for which $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ violates the conflict Constraint (3), that is, $y_i^* = y_j^* = 1$, then $\{i,j\}$ is a type 1 branch pair.*

Proof. Suppose that $(\boldsymbol{y}^*, \boldsymbol{z}^*)$ violates the conflict Constraint (3) for pair $\{i,j\} \in C$. Then $y_i^* = y_j^* = 1$. Because the subproblems for i and j contain Constraints (19), we have $z_{ij}^* = z_{ji}^* = 0$. Then $\{i,j\}$ is a type 1 pair. □

4. Experiments

In this section, we first introduce the social networks used in our computational study and explain how we generate our instances. Then we present the performance results of our branch-and-bound algorithm and its comparison with the mathematical models.

4.1. Data Sets and Instance Generation

Wi et al. (2009) use collaborative data from a research and development institute and form a social network of 45 researchers to test their genetic algorithm. Farasat and Nikolaev (2016) use existing social network data sets to test their heuristics, and the number of nodes in these networks varies from 15 to 500. By contrast, larger social networks are preferred in the knowledge discovery and data-mining literature. We follow the latter course and use the Internet Movie Database (IMDb) and Digital Bibliography & Library Project (DBLP) data sets in our computations.

IMDb is used by Anagnostopoulos et al. (2012) and Kargar and An (2011). We create our instances using the same part of the database used in the comparative study by Wang et al. (2015). The collaboration and skill information are provided by one of the authors on his website.[1] The nodes of the network are the actors who appeared in the movies from 2000 to 2002. There are 1,021 actors; that is, $|N| = 1{,}021$. The skills are the genres of the movies, and there are 27 skills. The social network contains an edge between actors i and j if they have worked together on a movie, and the weight of the edge equals the Jaccard distance, as explained in Section 2.

DBLP is the most common database used to generate instances for the TFP. It provides bibliographic information on papers published in major computer science journals and proceedings. We generate a social network from this database searching the papers published between 2010 and 2016. We narrow the search space by specifying journals and conferences. Because there is no keyword information for the papers in the database, we search the titles of the papers for some keywords and treat these keywords as the skills of the authors. There is an edge between two authors if they have at least two common papers in whole history. With this setting, we end up with 58 skills and a collaboration network, which has 12,855 nodes and 53,890 edges whose weights equal to the Jaccard distances. In both networks, we compute the shortest path lengths between all pairs, and if there is no path between i and j, we make the communication cost between i and j, p_{ij}, equal to a sufficiently large number. In Figure 4, to give an idea about the magnitudes and distribution of the communication costs, we plot the percentage of pairs whose distance is at most d for each network.

For both social networks, we created instances in the following way: The number of required skills m comes from the set $\{4, 6, 8, 10, 12, 14, 16, 18, 20\}$, and 100 random instances are generated for each m. The data sets and the instances used in the computational experiments are available in our Github repository.[2]

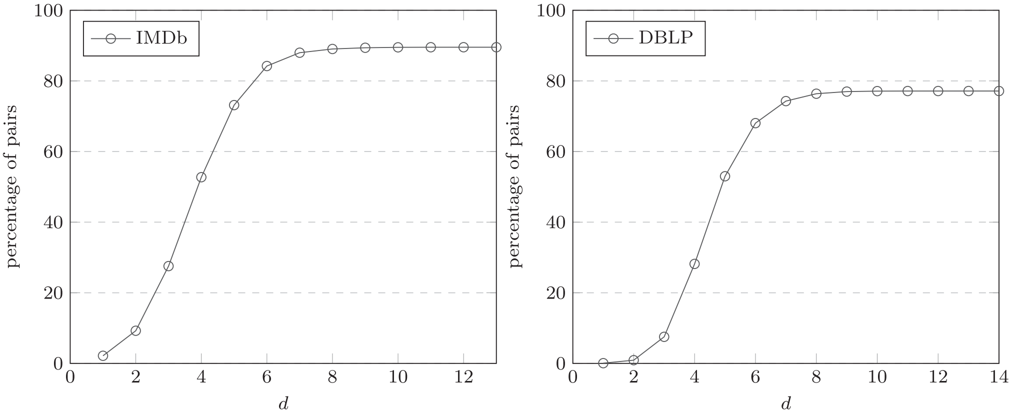
4.2. Computational Results

The mathematical models and the branch-and-bound algorithms are implemented in Java using CPLEX 12.7 and run on a personal computer with an Intel Core i7-6700HQ 2.6 GHz and 16 GB of random-access memory. All computational times reported in the tables are wall-clock times in seconds.

For each instance, it is sufficient to consider people who have one of the required skills. Therefore, we preprocess the input data and shrink the social network by removing people who do not possess any of the required skills. We call the remaining nodes in the network the *qualified* ones, and their number is denoted by *qno* in what follows. For the diameter-constrained version of the problem, we are able to reduce the network further by eliminating a person if he or she cannot cover all the skills together with the people who are at the most allowed diameter away from him or her. We do this elimination iteratively until there is no one to remove from the network. After this preprocessing, the network only involves people who are capable of forming a feasible team respecting the bound on the diameter. The number of candidates after preprocessing is denoted by *fno*.

In addition to the quadratic formulation (1), (2), (4) (denoted by QP); the mixed-integer formulation (2), (4)–(9) (denoted by MIP); and the branch-and-bound algorithm, we implemented a branch-and-cut algorithm

Figure 4. Percentage of Pairs Whose Shortest Distance Is at Most d in the IMDb (Left) and DBLP (Right) Networks



for the TFP-SD to overcome the memory problems for larger instances. In the mixed-integer formulation, the Constraints (6), (7), and (8) grow quadratically in the size of the problem. Because the objective coefficients are nonnegative in our instances, it is sufficient to use only Constraints (6), but even in this case, we have memory run-outs in the model-generation phase for large instances. When we use the original mixed-integer formulation without Constraints (7) and (8) and add Constraints (6) using the lazy cut pool (the constraints in this pool are only checked when an integer feasible solution is found and violated constraints are added to the formulation), a large number of lazy constraints are added, and consequently, this approach takes more time than solving the mixed-integer formulation directly. However, when we add the RLT Constraints (10), only a small number of lazy constraints are generated, and this improves the solution times. The cuts can also be applied at the fractional solutions by putting Constraints (6) to the user cut pool besides the lazy cut pool, but the computation times are longer in this case. Therefore, in our branch-and-cut implementation, we solve the mixed-integer programming formulation (2), (4), (5), (9), (10) by putting Constraints (6) to the lazy cut pool.

We report the average solution times of all solution procedures for the TFP-SD on the IMDb instances in Table 2. The averages are taken over 100 instances for each m. We present more detailed results for our branch-and-bound algorithm: *nodes* is the number of nodes evaluated, $lb - gap = 100(opt - lb)/opt$ and $ub - gap = 100(ub - opt)/opt$, where lb and ub are the lower and upper bounds at the root node, respectively, and opt is the optimal value. To see the strength of the linear programming relaxation of the mixed-integer formulation (2), (4)–(9), we also report $LP - gap = 100(opt - LP)/opt$, where LP is the optimal value of the linear programming relaxation. As can be seen in Table 2, the continuous relaxation is very weak.

The performances of the quadratic and mixed-integer formulations for the TFP-SD turn out to be very similar for the IMDb instances. On average, the optimal solution is reported within a minute or two by the solver with both mathematical models. When we compare these with the branch-and-bound algorithm, we clearly see the efficiency of the algorithm because it reaches the optimal solution six times faster than the models, on average. The instance with the longest solution time requires more than 1,300 seconds for both formulations, and it is solved in 19 seconds by the branch-and-bound algorithm. The longest time the branch-and-bound algorithm spends for an IMDb instance is actually 48.19 seconds. With the branch-and-cut algorithm, we are able to solve 98.6% of the instances within a minute, whereas this percentage is 78% for both the quadratic and mixed-integer formulations. When the number of required skills m is low, this method is as efficient as the branch-and-bound algorithm, but as m grows, the branch-and-bound algorithm outperforms the branch-and-cut algorithm as well. Analyzing the detailed results, we observe that for all instances with $m = 4$, the first incumbent found by the branch-and-bound algorithm is optimal. Although the quality degrades as the instances get larger, the initial upper bound is at most 1% away from the optimal in 93.55% of the instances.

In Table 3, we present the results for the TFP-SD on the DBLP instances. Because the DBLP network is a larger one, we could not obtain a solution from the mathematical models for most of the instances. Therefore, we only include the results for $m = 4, 6$, and 8 in this table to compare the performances. In general, we observe memory problems when the number of qualified people qno exceeds 2,100 and m is greater than 4. The column "*solved*" indicates the number of instances that can be solved to optimality out of 10. The average solution times are given for the instances solved. We see that with the mixed-integer and quadratic formulations we can only solve four instances with $m = 6$ and two instances with $m = 8$, whereas strengthening the model with RLT constraints and putting Constraints (6) to the lazy cut pool in the branch-and-cut framework enables us to solve more instances within less time. However,

Table 2. Results for the TFP-SD on the IMDb Instances

		QP	MIP		B&C	B&B			
m	*qno*	*time*	*time*	*LP − gap*	*time*	*time*	*nodes*	*lb − gap*	*ub − gap*
4	422.51	6.66	7.14	63.60	0.81	1.13	2.08	2.12	0.00
6	541.81	22.63	23.21	77.75	1.74	2.66	14.11	4.12	0.05
8	653.41	28.5	29.54	77.07	3.16	4.19	24.6	5.95	0.06
10	731.82	30.41	31.12	75.47	5.63	5.92	41.97	10.27	0.30
12	791.51	32.6	33.47	75.90	7.59	7.28	52.36	12.31	0.22
14	838.48	43.13	44.7	74.00	10.62	9.83	111.34	13.31	0.50
16	879.02	51.81	53.04	72.76	15.57	12.27	157.72	13.58	0.18
18	917.68	83.76	81.04	71.92	18.77	14.31	164.98	15.13	0.77
20	947.69	77.98	78.54	71.23	24.93	13.93	167.69	16.24	0.70

Table 3. Results for the TFP-SD on the DBLP Instances

		QP		MIP		B&C		B&B	
m	*qno*	*solved*	*time*	*solved*	*time*	*solved*	*time*	*solved*	*time*
4	1,650.5	10	343.04	10	359.75	10	53.95	10	9.84
6	2,239.80	4	336.79	4	352.96	10	142.59	10	20.65
8	2,896.50	2	386.29	2	2,508.42	8	279.39	10	36.56

eventually, this method also fails with memory problems as the size of instances increases. Furthermore, the performances of the mixed-integer and quadratic formulations which were very similar on the IMDb instances, start to differ as the problem size gets larger. The instances solved without memory problems are the same for both formulations, but the solution times of quadratic formulation are lower than those of mixed-integer. Having average solution times under a minute, the efficiency of the branch-and-bound algorithm is clearly seen in this table. Its longest solution time among these instances is actually 62.2 seconds.

We present detailed results of the branch-and-bound algorithm on the DBLP instances in Table 4. We also consider larger *m* values here. The column titled "*solved*" indicates the number of instances solved to optimality within a two-hour time limit over 100 instances for each *m*. The computational details presented in the table are for the instances that are solved within the time limit. We present the minimum, average, and maximum solution times for each *m* and also the standard deviation of these times under the columns titled "*min*", "*avg*", "*max*", and "*std*", respectively. The algorithm is able to solve all DBLP instances with $m = 4, 6, 8, 10$ within the limit, and actually, the highest solution time among these instances is around three minutes. When $m = 12$, there is only one instance that cannot be solved within two hours, and as *m* increases, we have few more. Among all, the algorithm is able to solve 43% of the instances in a minute and 97.8% of them in an hour. Similar to the results with the IMDb instances, the upper bound at the root node is very close to the optimal solution. In approximately 69% of the instances, this upper bound is at most 1% away from the optimal value.

In the IMDb and DBLP instances we use, because of the way skills are defined and assigned, it might be possible that closer nodes in the network have more skills in common. To investigate whether the performance of the algorithm is affected by such a possible correlation between distances and skills, we generated purely random skill matrices that do not have any connection with the distances. As before, we generate 100 instances for each *m* value. In Table 5, we present the results on these instances for the TFP-SD, where the new sets are denoted by IMDb^r and DBLP^r.

Each one of these instances is solved within the two-hour time limit by the branch-and-bound algorithm. The solution times of the random IMDb^r instances are higher than those of the original ones presented in Table 2 when we compare the corresponding rows for each *m*. However, we must observe that the average number of qualified people *qno* for each *m* is also higher in the new set of instances.

In the rest of this section, we present the results of the computational experiments for the diameter-constrained version of the problem, DC-TFP-SD. In what follows, QP represents the quadratic formulation (1)–(4), MIP represents the mixed-integer formulation (2)–(9), and B&B represents the branch-and-bound algorithm developed for the DC-TFP-SD. For the IMDb instances, we first found the optimal diameters. In Table 6, we present the solution times for

Table 4. Detailed Results of the Branch-and-Bound Algorithm for the TFP-SD on the DBLP Instances

			time						
m	*qno*	*solved*	*min*	*avg*	*max*	*std*	*nodes*	*lb − gap*	*ub − gap*
4	1,540.22	100	0.48	8.59	42.43	9.02	20.08	4.97	0.05
6	2,255.9	100	1.53	20.68	67.55	13.59	30.54	6.47	0.27
8	2,963.26	100	1.88	37.69	107.27	21.57	110.52	8.16	0.48
10	3,604.4	100	6.75	59.86	191.79	17.10	239.10	7.99	0.69
12	4,189.49	99	20.65	89.41	275.92	49.05	480.52	8.56	0.89
14	4,789.13	99	35.60	249.25	4,921.88	633.18	3,374.63	8.79	0.89
16	5,298.52	99	47.47	274.76	3,571.15	482.30	3,099.22	8.62	0.66
18	5,857.6	97	66.48	482.83	4,743.17	707.73	5,637.57	9.25	0.76
20	6,412.48	91	114.81	680.89	4,998.51	1,030.91	6,439.47	9.32	0.82

Table 5. Results of Branch-and-Bound Algorithm for the TFP-SD on IMDb^r and DBLP^r: The IMDb and DBLP Instances with Randomly Generated Skill Matrices

	IMDb^r					DBLP^r				
m	*qno*	*min*	*avg*	*max*	*std*	*qno*	*min*	*avg*	*max*	*std*
4	531.91	0.50	1.95	3.23	0.59	1,602.76	1.30	11.14	26.41	5.32
6	693.47	1.43	3.96	10.89	1.13	2,247.75	5.25	21.67	41.12	7.23
8	794.86	2.79	5.44	15.22	1.30	2,878.79	9.23	34.54	74.43	10.18
10	863.80	4.44	6.91	9.75	0.85	3,485.32	16.41	55.12	152.94	19.98
12	912.65	6.55	9.40	26.72	3.08	4,065.39	30.42	80.16	161.53	25.24
14	949.29	8.99	12.86	62.74	6.10	4,587.13	55.27	142.26	866.07	95.97
16	971.96	11.87	18.30	88.25	9.19	5,112.02	71.10	248.54	1,239.47	214.24
18	986.62	15.23	30.20	215.19	28.92	5,591.90	89.30	396.69	1,735.91	362.95
20	997.61	20.20	48.58	421.12	62.72	6,018.28	115.99	580.47	3,718.81	704.72

the DC-TFP-SD, where we use the optimal diameter values as the diameter bounds. Because of preprocessing, the network is reduced significantly (we remind readers that *fno* is the number of candidates after preprocessing), and therefore, the solutions times of all methods are very small. For example, when $m = 20$, the network consists of more than 900 qualified people, on average, but the number of candidates reduces to approximately 200 people when we exclude the people who cannot build a team respecting the bound on the diameter. Therefore, the solution times are only one or two seconds for all methods.

We continued the experiments of the DC-TFP-SD with the IMDb instances with varying bounds on the diameter. In Table 7, we present the results with $D = 2$ and 3. Under the column "*feas*," the number of feasible instances is given over 100 instances for each *m*. In the IMDb instances, the optimal diameter is usually less than two, and therefore, the number of candidates that remain after preprocessing is greater with $D = 2, 3$ than with D taken as the optimal diameter. Thus, as the bound on the diameter increases, so does the size of the network, and we start to observe differences in the performances of the solution methods. When the bound on diameter is taken as its optimal value, all solution procedures are able to find optimal solutions within one or two seconds. When we take the bound as two and three, the solution times of MIP and QP become 15 seconds, on average, whereas it is still a couple of seconds for the branch-and-bound algorithm. To be more specific, the maximum solution times of MIP, QP, and the branch-and-bound algorithm with $D = 2$ are 60, 43, and 12 seconds, and with $D = 3$, they are 197, 189, and 23 seconds, respectively.

Table 6. Results for the DC-TFP-SD on the IMDb Instances Where the Bound on the Diameter Is Taken as the Optimal Diameter

			QP	MIP	B&B			
m	*qno*	*fno*	*time*	*time*	*time*	*nodes*	*lb − gap*	*ub − gap*
4	422.51	8.69	0.01	0.01	0.02	0.67	0.45	0.00
6	541.81	20.00	0.02	0.02	0.09	4.57	0.64	0.05
8	653.41	41.52	0.06	0.09	0.30	6.54	1.36	0.00
10	731.82	69.77	0.13	0.18	0.28	13.77	2.01	0.01
12	791.51	91.99	0.24	0.31	0.44	22.31	2.61	0.12
14	838.48	119.16	0.49	0.56	0.65	22.04	2.70	0.04
16	879.02	152.62	0.89	0.98	1.10	45.02	3.67	0.06
18	917.68	178.94	1.18	1.35	1.49	69.81	3.93	0.08
20	947.69	216.11	1.80	2.07	2.20	104.34	4.71	0.09

For the DBLP instances, we used $D = 1, 2, 3$, and 4 as the bounds on the diameter. To be able to compare the solution procedures, we first present the results of the first 10 instances with $m = 4, 6, 8$, and 10 for $D = 2$ and $D = 3$ in Table 8. For these instances, average solution time of MIP is a few minutes and usually less than that of QP. Nevertheless, QP is able to solve all these instances, whereas we encounter memory errors with MIP when $D = 3$ and *m* exceeds six. The branch-and-bound algorithm, by contrast, is able to solve each of these instances in under 30 seconds.

In Table 9, we present the results for all DBLP instances using the branch-and-bound algorithm for the DC-TFP-SD with $D = 1, 2, 3, 4$. The averages of solution times are taken over the instances that are

Table 7. Results for the DC-TFP-SD on the IMDb Instances

	$D = 2$					$D = 3$				
m	*feas*	*fno*	MIP	QP	B&B	*feas*	*fno*	MIP	QP	B&B
4	94	250.01	8.26	5.38	0.47	97	317.16	15.24	7.86	0.72
6	88	266.10	10.94	7.80	0.76	92	375.60	25.60	16.67	1.47
8	79	265.19	11.75	8.48	0.95	87	402.48	28.89	19.74	1.89
10	66	279.82	13.10	9.47	1.28	79	427.18	36.49	20.97	2.43
12	60	255.93	13.11	8.97	1.31	74	424.51	34.69	20.47	2.71
14	48	208.94	11.65	7.59	1.28	68	389.79	30.79	17.85	2.85
16	36	208.03	10.97	7.84	1.30	60	382.97	29.96	17.21	2.76
18	24	165.79	10.72	6.73	1.49	54	353.80	23.37	15.22	3.22
20	14	192.79	16.30	8.42	1.47	45	349.62	21.74	16.03	3.81

Table 8. Results for the DC-TFP-SD on the DBLP Instances

		$D = 2$							$D = 3$					
		MIP		QP		B&B			MIP		QP		B&B	
m	*feas*	*solved*	*time*	*solved*	*time*	*solved*	*time*	*feas*	*solved*	*time*	*solved*	*time*	*solved*	*time*
4	9	9	64.48	9	98.61	9	1.47	10	10	303.60	10	340.22	10	5.15
6	5	5	37.95	5	48.09	5	1.43	10	10	318.63	10	285.92	10	6.87
8	3	3	81.08	3	113.66	3	3.36	10	9	328.54	10	535.28	10	10.04
10	1	1	244.01	1	415.34	1	22.41	8	7	161.45	8	661.48	8	9.78

Table 9. Results of the Branch-and-Bound Algorithm for the DC-TFP-SD on the DBLP Instances

	$D = 1$			$D = 2$			$D = 3$			$D = 4$		
m	*feas*	*solved*	*time*	*feas*	*solved*	*time*	*feas*	*solved*	*time*	*feas*	*solved*	*time*
4	35	35	0.08	83	83	1.87	99	99	4.06	100	100	6.60
6	7	7	0.02	64	64	1.48	97	97	7.51	100	100	15.11
8	1	1	0.03	36	36	6.60	92	92	12.26	100	100	27.02
10	0	0	0	21	21	15.24	82	82	22.42	98	98	35.14
12	0	0	0	14	14	10.64	78	78	124.95	96	96	51.77
14	0	0	0	9	9	9.76	68	67	56.31	94	94	131.83
16	0	0	0	2	2	5.51	56	54	139.12	93	91	176.08
18	0	0	0	2	2	3.57	49	48	267.20	90	87	297.41
20	0	0	0	0	0	0.00	38	35	187.83	87	83	366.99

solved within two hours of time limit, and there are only 16 unsolved instances among 1,791 feasible ones. The algorithm is able to solve 79% and 96% of the feasible instances within 1 and 10 minutes, respectively.

5. Conclusion

In this study, we formulated the TFP as a quadratic set covering problem with packing constraints and developed a novel branch-and-bound algorithm to solve it. The algorithm uses a relaxation that can be solved by solving a series of linear set covering problems and a different branching rule compared with existing branch-and-bound methods for quadratic 0–1 optimization problems. Our computational experiments on TFP instances show that the algorithm is capable of solving large sizes that are intractable for the solver. The same approach can be used to solve other binary quadratic problems, but success depends, among other things, on how quickly the relaxation (the corresponding 0–1 linear problems) can be solved.

In terms of application, this work can be extended in several ways. First, the communication cost may be quantified with respect to tasks, in which case the problem also requires assigning people to tasks. Second, the uncertainty in the communication costs can be incorporated into the decision-making process using robust optimization and stochastic programming. This can be done in a single-stage setting where the worst case or expected communication cost can be minimized, or it can be done in a multistage setting where decisions can be updated over time to improve the performance of the team.

Acknowledgments

Part of the research of H. Yaman was carried out in the Department of Industrial Engineering at Bilkent University.

Endnotes

[1] See http://home.cse.ust.hk/faculty/wilfred/wangxinyu/.

[2] See https://github.com/nihalberktas/TFP-data.

References

Adams WP, Sherali HD (1986) A tight linearization and an algorithm for zero-one quadratic programming problems. *Management Sci.* 32(10):1274–1290.

Adams WP, Guignard M, Hahn PM, Hightower WL (2007) A level-2 reformulation–linearization technique bound for the quadratic assignment problem. *Eur. J. Oper. Res.* 180(3):983–996.

Agustín-Blas LE, Salcedo-Sanz S, Ortiz-García EG, Portilla-Figueras A, Pérez-Bellido ÁM, Jiménez-Fernández S (2011) Team formation based on group technology: A hybrid grouping genetic algorithm approach. *Comput. Oper. Res.* 38(2):484–495.

Anagnostopoulos A, Becchetti L, Castillo C, Gionis A, Leonardi S (2012) Online team formation in social networks. Mille A, ed. *Proc. 21st Internat. Conf. World Wide Web* (ACM, New York), 839–848.

Avgerinos E, Gokpinar B (2016) Team familiarity and productivity in cardiac surgery operations: The effect of dispersion, bottlenecks, and task complexity. *Manufacturing Service Oper. Management* 19(1):19–35.

Baykasoglu A, Dereli T, Das S (2007) Project team selection using fuzzy optimization approach. *Cybern. Systems Internat. J.* 38(2):155–185.

Bazaraa MS, Goode JJ (1975) A cutting-plane algorithm for the quadratic set-covering problem. *Oper. Res.* 23(1):150–158.

Bergman D (2019) An exact algorithm for the quadratic multiknapsack problem with an application to event seating. *INFORMS J. Comput.* 31(3):477–492.

Bhowmik A, Borkar VS, Garg D, Pallan M (2014) Submodularity in team formation problem. Zaki M, Kamath C, Banerjee A, Parthasarathy A, Tan PN, Obradovic Z, eds. *Proc. 2014 SIAM Internat. Conf. Data Mining* (SIAM, Philadelphia), 893–901.

Billionnet A, Calmels F (1996) Linear programming for the 0–1 quadratic knapsack problem. *Eur. J. Oper. Res.* 92(2):310–325.

Boon BH, Sierksma G (2003) Team formation: Matching quality supply and quality demand. *Eur. J. Oper. Res.* 148(2):277–292.

Caprara A, Pisinger D, Toth P (1999) Exact solution of the quadratic knapsack problem. *INFORMS J. Comput.* 11(2):125–137.

Centric Digital (2016) What is TAAS (team as a service) and why is it becoming so popular? Retrieved April 6, 2017, https://centricdigital.com/blog/digital-trends/what-is-team-as-a-service/.

Chen SJ, Lin L (2004) Modeling team member characteristics for the formation of a multifunctional team in concurrent engineering. *IEEE Trans. Engrg. Management* 51(2):111–124.

de Klerk E, Sotirov R, Truetsch U (2015) A new semidefinite programming relaxation for the quadratic assignment problem and its computational perspectives. *INFORMS J. Comput.* 27(2): 378–391.

Dorn C, Dustdar S (2010) Composing near-optimal expert teams: a trade-off between skills and connectivity. Meersman R, Dillon T, Herrero P, eds. *OTM Confederated Internat. Conf. Move Meaningful Internet Systems* (Springer, New York), 472–489.

Escoffier B, Hammer PL (2007) Approximation of the quadratic set covering problem. *Discrete Optim.* 4(3–4):378–386.

Espinosa JA, Slaughter SA, Kraut RE, Herbsleb JD (2007) Familiarity, complexity, and team performance in geographically distributed software development. *Organ. Sci.* 18(4):613–630.

Farasat A, Nikolaev AG (2016) Social structure optimization in team formation. *Comput. Oper. Res.* 74:127–142.

Fitzpatrick EL, Askin RG (2005) Forming effective worker teams with multi-functional skill requirements. *Comput. Industrial Engrg.* 48(3):593–608.

Fomeni FD, Kaparis K, Letchford AN (2014) A cut-and-branch algorithm for the quadratic knapsack problem. Technical report, Lancaster University Management School, Lancaster, UK.

Fortet R (1960) Applications de l'algebre de boole en recherche opérationelle. *Rev. Française Recherche Opérationelle* 4(14):17–26.

Gajewar A, Sarma AD (2012) Multi-skill collaborative teams based on densest subgraphs. Ghosh J, Liu H, Davidson I, Domeniconi C, Kamath C, eds. *Proc. 2012 SIAM Internat. Conf. Data Mining* (SIAM, Philadelphia).

Guimarães DA, da Cunha AS, Pereira DL (2020) Semidefinite programming lower bounds and branch-and-bound algorithms for the quadratic minimum spanning tree problem. *Eur. J. Oper. Res.* 280(1):46–58.

Gutiérrez JH, Astudillo CA, Ballesteros-Pérez P, Mora-Melià D, Candia-Véjar A (2016) The multiple team formation problem using sociometry. *Comput. Oper. Res.* 75:150–162.

Hahn PM, Zhu YR, Guignard M, Hightower WL, Saltzman MJ (2012) A level-3 reformulation-linearization technique-based bound for the quadratic assignment problem. *INFORMS J. Comput.* 24(2): 202–209.

Hoegl M, Gemuenden HG (2001) Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence. *Organ. Sci.* 12(4):435–449.

Huckman RS, Staats BR, Upton DM (2009) Team familiarity, role experience, and performance: Evidence from Indian software services. *Management Sci.* 55(1):85–100.

Jaccard P (1912) The distribution of the flora in the alpine zone. 1. *New Phytology* 11(2):37–50.

Jones R (2005) Working Virtually: Challenges of Virtual Teams. Jones R, Oyung R, Pace L, eds. *Challenges of Virtual Teams* (IGI Global, Hershey, PA).

Kargar M, An A (2011) Discovering top-k teams of experts with/without a leader in social networks. Berendt B, de Vries AP, Fan W, Macdonald C, Ruthven IG, eds. *Proc. 20th ACM Internat. Conf. Inform. Knowledge Management* (ACM, New York), 985–994.

Kargar M, An A, Zihayat M (2012) Efficient bi-objective team formation in social networks. Flach PA, De Bie T, Cristiani N, eds. *Joint Eur. Conf. Machine Learn. Knowledge Discovery Databases* (Springer, New York), 483–498.

Lappas T, Liu K, Terzi E (2009) Finding a team of experts in social networks. Elder J, Fogelman FS, Flach PA, Zaki MJ, eds. *Proc. 15th ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining* (ACM, New York), 467–476.

Loiola EM, de Abreu NMM, Boaventura-Netto PO, Hahn P, Querido T (2007) A survey for the quadratic assignment problem. *Eur. J. Oper. Res.* 176(2):657–690.

Majumder A, Datta S, Naidu K (2012) Capacitated team formation problem on social networks. Yang Q, Agarwal D, Pei, J, eds. *Proc. 18th ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining* (ACM, New York), 1005–1013.

Martinelli R, Contardo C (2015) Exact and heuristic algorithms for capacitated vehicle routing problems with quadratic costs structure. *INFORMS J. Comput.* 27(4):658–676.

Mittelmann H, Peng J (2010) Estimating bounds for quadratic assignment problems associated with hamming and manhattan distance matrices based on semidefinite programming. *SIAM J. Optim.* 20(6): 3408–3426.

Pandey P, Punnen AP (2017) On a linearization technique for solving the quadratic set covering problem and variations. *Optim. Lett.* 11(7):1357–1370.

Pisinger WD, Rasmussen AB, Sandvik R (2007) Solution of large quadratic knapsack problems through aggressive reduction. *INFORMS J. Comput.* 19(2):280–290.

Povh J, Rendl F (2009) Copositive and semidefinite relaxations of the quadratic assignment problem. *Discrete Optim.* 6(3):231–241.

Punnen AP, Pandey P, Friesen M (2019) Representations of quadratic combinatorial optimization problems: A case study using quadratic set covering and quadratic knapsack problems. *Comput. Oper. Res.* 112:104769.

Rodrigues C, Quadri D, Michelon P, Gueye S (2012) 0-1 quadratic knapsack problems: an exact approach based on a t-linearization. *SIAM J. Optim.* 22(4):1449–1468.

Saxena R, Arora S (1997) A linearization technique for solving the quadratic set covering problem. *Optim.* 39(1):33–42.

Wang X, Zhao Z, Ng W (2015) A comparative study of team formation in social networks. Renz M, Shahabi C, Zhou X, Cheema MA, eds. *Internat. Conf. Database Systems Advanced Applications* (Springer, New York), 389–404.

Wi H, Oh S, Mun J, Jung M (2009) A team formation model based on knowledge and collaboration. *Expert Systems Appl.* 36(5): 9121–9134.

Zakarian A, Kusiak A (1999) Forming teams an analytical approach. *IIE Trans.* 31(1):85–97.

Zhang L, Zhang X (2013) Multi-objective team formation optimization for new product development. *Comput. Industrial Engrg.* 64(3):804–811.