Recoverable Team Formation: Building Teams Resilient to Change

Emir Demirović
School of Computing and Information Systems, University of Melbourne, Australia
emir.demirovic@unimelb.edu.au

Tenda Okimoto
Kobe University, Kobe, Japan
tenda@maritime.kobe-u.ac.jp

Nicolas Schwind
National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
nicholas-schwind@aist.go.jp

Katsumi Inoue
National Institute of Informatics, Tokyo, Japan
inoue@nii.ac.jp

ABSTRACT
Team formation consists in finding the least expensive team of agents such that a certain set of skills is covered. In this paper, we formally introduce recoverable team formation (RTF), a generalization of the above problem, by taking into account the dynamic nature of the environment, e.g., after a team has been formed, agents may unexpectedly become unavailable due to failure or illness. We analyze the computational complexity of RTF, provide both complete and heuristic algorithms, and empirically evaluate their performance. Furthermore, we demonstrate that RTF generalizes robust team formation, where the task is to build a team capable of covering all required skills even after any $k$ agents are removed. Despite the high complexity of forming a recoverable team, we argue that recoverability is a crucial feature, and experimentally show that it is more appropriate for some applications than robustness.

ACM Reference Format:

1 INTRODUCTION
Team formation (TF) is the problem of selecting a team of agents with minimum cost such that a certain set of skills is covered\(^1\). This is an important problem in multi-agent systems and has been studied in the contexts of RoboCup rescue teams [17], unmanned aerial vehicle operations [9], social networks [16, 18], online football prediction games [19], among others. Moreover, TF is closely related to set covering and hitting sets [8], well known NP-complete problems. More precisely, we are given a set of agents and a set of skills to be covered. Each agent possesses a subset of skills and is associated with a cost. A subset of agents (referred to as a team) is said to be efficient if for every required skill there is at least one agent in the team possessing that skill. The goal is to select an efficient team with minimum cost.

Once a team has been formed, we can expect it to undergo changes with time. For example, agents may unexpectedly become unavailable due to failure or illness, possibly making the team no longer efficient. This results in additional expenses, inconveniences, and in some applications in complete system failure. Therefore, in addition to the team’s cost, it can be of crucial importance to analyze its ability to react to changes: how resilient the team is.

One aspect of resilience for TF has been introduced in [23], namely the concept of robustness. A team is said to be $k$-robust if the team remains efficient even after $k$ agents are removed from it. Robustness is clearly a desirable property as it allows the underlying system to remain functional after an unfortunate change happens. Interestingly, the computational complexity of the robust TF problem is the same as the standard TF problem: it is also NP-complete.

However, considering robustness alone may have the following drawback: while highly robust teams can easily withstand unfortunate changes, this is achieved by introducing a high degree of redundancy to over-prepare for the future, which may result in prohibitively expensive teams. To circumvent this negative aspect while accounting for unfortunate changes, we introduce the concept of recoverability for TF. This concept has been considered in the literature as another main feature of resilience [4, 14, 26], as the capacity to cope with unanticipated dangers after they have become effectively manifested on the system, and described as the ability of a system to return to an equilibrium state after some temporary disturbance. Indeed, a resilient system must find an appropriate balance between robustness (sometimes called also resistance [26], see our related work section) and recoverability. In TF, we introduce recoverability by considering an additional cost which indicates the expense that might need to be incurred to restore the team’s efficiency if $k$ agents are removed from the team.

Considering recoverability allows for more flexibility in TF: whereas robustness underlies a fully proactive strategy where we over-prepare for every possible future event, recoverability provides a balance between proactive and reactive approaches where we analyze both the initial and the next-step cost that will be needed if an undesirable event takes place and effectively damages the team.

\(^1\)In the literature sometimes the term task [23] is used instead of skill, but the problem remains the same.
Example 1.1. Consider the task of developing health care facilities on an earthquake-risk area given in Fig. 1. The area consists of a 3×3 grid where each subarea (i.e., each “box” in the grid) (i) accommodates citizens who must have access to a facility, and (ii) can itself host a facility. Three types of facilities are available for construction: hospitals, clinics, and rescue centers. The following assumptions are considered: only one hospital can be erected, but the other types of facilities can be deployed at several places. As hospitals and clinics need time to be built, only rescue centers may be built on short notice in case of an emergency. A cost and a service range is associated with each facility. Each hospital, clinic, and rescue center is respectively associated with a deployment cost of 80, 20 and 20 \(^2\) and a service range of 2, 1 and 0; that is, each subarea of coordinates \((x, y)\) can access to a facility \(F\) with range \(r_F\) located at coordinates \((x_F, y_F)\) if \(|x - x_F| + |y - y_F| \leq r_F\). The goal is to ensure that every subarea has access to at least one facility.

In this example, “agents” refer to facilities and “teams” refer to sets of facilities. Fig. 1(a) and 1(b) depict two efficient teams \(T_1\) and \(T_2\); each of them allows any subarea to access to a facility. \(T_1\) has a deployment cost of 80 and \(T_2\) is an optimal one since it has a deployment cost of 60 which is minimal among all efficient teams. Now assume that a disaster damages one of the subareas of the grid, effectively rendering the facility at the corresponding location as nonfunctional. In this case, none of the teams \(T_1, T_2\) are satisfactory. Indeed, in \(T_1\) if the hospital becomes nonfunctional, the whole area is left without medical care and nine rescue centers — the only type of facility which can be deployed in an emergency situation — need to be deployed, at a prohibitive 1-recovery cost of 180. In \(T_2\), if one of the clinics is damaged, then in the worst case (if the clinic at coordinates \((2, 3)\) collapses) four subareas become out of reach of the remaining clinics, which requires building four rescue centers, at a 1-recovery cost of 80. The 1-overall cost (deployment cost + 1-recovery cost) invested may then be \(80 + 180 = 260\) for \(T_1\), and \(60 + 80 = 140\) for \(T_2\). The team \(T_3\) (cf. Fig. 1(c)) has a deployment cost of 140. It is not only efficient, but if one of the facility collapses in case of disaster, any subarea of the grid is still under the reach of one of the remaining facilities. That is, \(T_3\) is 1-robust. Moreover, it is the optimal 1-robust team. Deploying \(T_3\) is appropriate when it is of utmost importance to provide service to every subarea at all times. However, such a solution might still be seen as infeasible due to budget limitations. Instead, one might consider deploying \(T_4\) (cf. Fig. 1(d)) which consists of four clinics. \(T_4\) has a deployment cost of 80 which is considerably less than the cost of \(T_3\). In the unfortunate event where one facility is lost to an unexpected event, one can check that in any of the four cases only one rescue center needs to be deployed (Fig. 1(e) depicts an example). Thus, in the worst case, deploying \(T_4\) requires an additional cost of 20, assuming one of the facility may be lost. One can see in this example that \(T_4\), the “recoverable” alternative, is more satisfactory than the one of \(T_3\) in terms of overall cost: \(80 + 20 = 100\) for \(T_4\) in the worst case (deployment + 1-recovery cost), versus 140 for deploying \(T_3\).

Our contributions are as follows. We define the recoverable TF problem and prove its computational complexity to be \(\text{NP}\)-complete. Given that the standard and robust TF problems are both “only” \(\text{NP}\)-complete problems, conventional techniques for TF can no longer be applied. To deal with the complexity shift, we introduce a novel algorithm with two key components: nonlinear cuts to prune teams with suboptimal substructures, and a search strategy to implicitly simplify the cuts. In addition, we propose a heuristic cut to provide a trade-off between computational time and solution quality. Lastly, we experimentally show that even though the recoverable TF problem is more complex than the robust TF problem, we are able to compute optimal solutions for a number of benchmarks. Interestingly, our experiments show that recoverable teams are more satisfactory than robust ones in terms of overall cost.

For space reasons, most proofs are omitted. An extended version of the paper containing all the proofs is available from https://staff.aist.go.jp/nicolas-schwind/AAMAS2018.pdf.

2 (ROBUST) TEAM FORMATION

We provide the main definitions and theorems for (robust) team formation (TF). More details can be found in [23].

Definition 2.1. (TF Problem Description) A TF problem description is a tuple \(\text{TF} = (A, S, f, \alpha)\) where \(A = \{a_1, a_2, ..., a_n\}\) is a set of agents, \(S = \{s_1, s_2, ..., s_m\}\) is a set of skills, \(f : 2^A \to \mathbb{N}\) is a cost function, and \(\alpha\) is a mapping from \(A\) to \(2^S\). \(T \subseteq A\) is called a team.

It is assumed that the cost of a team \(T\) is given by the sum of the costs of each agent, i.e., \(f(T) = \sum_{a_i \in T} f(a_i)\).

Definition 2.2. Team \(T\) is said to be \(c\)-costly if \(f(T) \leq c\).

Definition 2.3. Team \(T\) is efficient if \(S = \bigcup_{a_i \in T} \alpha(a_i)\).

Figure 1: Illustrative example: efficient, robust and recoverable deployment strategies of health care facilities.

---

These values are arbitrary and only reflect the relative cost of deployment between the different types of facilities.

---
The decision problem for TF (labeled Skill Efficient Team Formation - SETF), given a TF problem description and a nonnegative integer \( c \) as input, asks if there exists a team \( T \subseteq A \) such that \( T \) is \( c \)-costly and efficient.

**Theorem 2.4.** [23] SETF is NP-complete.

**Definition 2.5.** [23] A team \( T \) is said to be \( k \)-robust if for every set of agents \( T' \subseteq T \) such that \( |T'| \leq k \), the team \( T \setminus T' \) is efficient.

Note that robustness generalizes efficiency (for \( k > 0 \)). The decision problem for robustness (labeled Skill Oriented Robust Team Formation - SORTF), given a TF problem description and integers \( c \) and \( k \), asks if there exists a team \( T \subseteq A \) such that \( T \) is \( c \)-costly and \( k \)-robust.

**Theorem 2.6.** [23] SORTF is NP-complete.

The optimization variant of SORTF asks for a robust team with minimum cost.

## 3 RECOVERABLE TEAM FORMATION

We now introduce recoverable team formation (RTF), a generalization of the standard TF problem, where in addition to the team’s initial cost we consider the recovery cost that represents the cost required to restore efficiency after any \( k \) agents are removed; efficiency is restored by adding agents that have not been previously selected.

**Definition 3.1 (Recoverable Team Formation Problem Description).** An RTF problem description is a tuple \( \langle A, S, f, \alpha, h \rangle \) where \( A = \{a_1, a_2, \ldots, a_n\} \) is a set of agents, \( S = \{s_1, s_2, \ldots, s_m\} \) is a set of skills, \( f : 2^A \rightarrow \mathbb{N} \cup \{+\infty\} \) is the deployment cost function, \( \alpha \) is a mapping from \( A \) to \( 2^S \), and \( h : 2^A \rightarrow \mathbb{N} \cup \{+\infty\} \) is the recovery cost function.

As in the robust TF setting, we assume that the cost functions \( f \) and \( h \) satisfy \( f(T) = \sum_{a_i \in T} f(a_i) \) and \( h(T) = \sum_{a_i \in T} h(a_i) \) for each team \( T \). The function \( f \) is used to characterize the cost of deploying a given team in a “standard” or “initial” situation, as in the standard and robust TF settings. The function \( h \), however, is used to define the cost of deploying a rescue team in an “emergency” situation, i.e., to restore the team’s efficiency after a set of agents has been removed from the initially formed team.

We are now ready to introduce our property of interest, i.e., team recoverability. Beforehand, let us introduce the notion of “k-recovery cost” of a given team, a key additional component in RTF. Given a team \( T \) and a number \( k \), the \( k \)-recovery cost of \( T \) corresponds to the minimum recovery cost necessary to restore the efficiency loss of \( T \) after removing \( k \) agents from it in the worst case. Formally:

**Definition 3.2 (Recovery and Overall Cost).** Let \( \langle A, S, f, \alpha, h \rangle \) be an RTF problem description. Given a team \( T \subseteq A \) and a non-negative integer \( k \), the \( k \)-recovery cost of \( T \), denoted \( rc(T, k) \), is defined as

\[
rc(T, k) = \max_{T' \subseteq T, |T'| = k} rcS(T, T'),
\]

where \( rcS(T, T') \) corresponds to the cost involved in recovering the team \( T \) after being deprived of \( T' \) from it, i.e.,

\[
rcS(T, T') = \min_{T' \subseteq T} \{ h(T_{rec}) \mid T_{rec} \text{ is efficient} \}
\]

The \( k \)-overall cost, denoted \( oc(T, k) \), is defined as

\[
oc(T, k) = f(T) + rc(T, k).
\]

The \( k \)-recovery cost represents the amount that needs to be paid in the worst case when \( k \) agents are removed from the team. While a robust team is required to remain efficient after \( k \) agents are removed, a recoverable team aims for a low \( k \)-overall cost, having a relatively “harmless” recovery cost if \( k \) agents are lost.

**Definition 3.3 (Team Recoverability).** Given an RTF problem description \( \langle A, S, f, \alpha, h \rangle \) and a team \( T \), \( T \) is \( (k, c_2) \)-recoverable if \( rc(T, k) \leq c_2 \), that is, if for every \( T' \subseteq T \) such that \( |T'| \leq k \), there exists \( T_{rec} \subseteq (A \setminus T) \) such that the team \( (T \setminus T') \cup T_{rec} \) is an efficient team and \( h(T_{rec}) \leq c_2 \). Additionally, if \( f(T) \leq c_1 \), then \( T \) is said to be \( (c_1, k, c_2) \)-recoverable.

Interestingly, our notion of recoverability can be viewed as a generalization of the notion of robustness, i.e., both notions coincide when restoring the team’s initial efficiency must be done with no \( k \)-recovery cost. Indeed:

**Proposition 3.4.** Let \( \langle A, S, f, \alpha, h \rangle \) be a RTF problem description such that \( h \) is defined for each \( a_i \in A \) as \( h(a_i) \) \( > 0 \), then for every efficient team \( T \subseteq A \) and every \( k \geq 0 \), \( T \) is \( k \)-robust and \( c \)-costly if and only if \( T \) is \( (c, k, 0) \)-recoverable.

**Example 3.5 (continued).** Let us formalize the health care facility location example provided in the introduction. For each facility of type \( t \in \{H, C, R\} \) (where \( H, C, R \) respectively stand for hospital, clinic and rescue center) and each pair of coordinates \((x, y)\) where the facility can be deployed, we introduce an agent denoted \( a_{xy}^t \). Recall that clinics and rescue centers can be placed anywhere, but a hospital can be deployed only at coordinates \((2, 2)\). That is, the set of agents \( A \) is defined as \( A = \{a_{xy}^t \mid t \in \{C, R\}, x, y \in \{1, 2, 3\}\} \cup \{a_x^H \} \). A skill is associated with each subarea of the grid, i.e., each pair of coordinates \((i, j), i \in \{1, 2, 3\}\) is associated with the skill denoted \( s_{ij} \) and the set of skills \( S \) is defined as \( S = \{s_{ij} \mid i, j \in \{1, 2, 3\}\} \).

An agent \( a_{xy}^t \in A \) has a skill \( s_{ij} \in S \) if the subarea represented by \( s_{ij} \) is within the reach of the facility of type \( t \) located at coordinates \((x, y)\). Hence, for each \( a_{xy}^t \in A \), \( a_{xy}^t(s_{ij}) = \{s_{ij} \in S \mid |x - i| + |y - j| \leq range(t)\} \), where \( range(H) = 2 \), \( range(C) = 1 \) and \( range(R) = 0 \). In addition, \( f(H) = 80 \), \( f(C) = f(R) = 20 \) (it costs 80 to deploy a hospital and 20 to deploy a clinic or a rescue center). Lastly, \( h(H) = h(C) = +\infty \) and \( h(R) = 20 \) (only rescue centers are available for construction on short notice, when needed after a disaster occurs).

The four teams depicted in our introductory example (cf. Fig. 1) are defined as \( T_1 = \{a_{x}^{H}, a_{y}^{C}, a_{z}^{R}\} \), \( T_2 = \{a_{x}^{H}, a_{y}^{C}, a_{z}^{R}\} \), \( T_3 = T_1 \cup T_2 \), and \( T_4 = \{a_{x}^{H}, a_{y}^{C}, a_{z}^{R}\} \). Table 1 provides for each team \( T_i \) its deployment cost and its \( k \)-recovery cost for \( k = \{1, \ldots, 4\} \), i.e., the cost of recovering the team’s efficiency in the worst case after losing \( k \) agents. \( T_1 \) and \( T_2 \) are efficient teams and \( T_3 \) is optimal in the sense that \( f(T_2) \) is minimal among all efficient teams. \( T_3 \) is 1-robust and is optimal as well (w.r.t. \( f \)). \( T_4 \) is, for instance, \((80, 1, 20)\)-recoverable. It can be seen that while \( T_4 \) appears to be initially more costly than \( T_2 \) and is not 1-robust (as \( T_3 \) is), it exhibits an interesting compromise in terms of \( k \)-recovery costs: for each \( k \in \{1, \ldots, 3\} \), \( rc(T_4, k) \) is strictly lower than the other teams, which means that
recovering $T_4$ in the case of any number of agent loss up to three is guaranteed to be less costly than the optimal efficient and robust teams $T_2$ and $T_3$. In addition, it can be checked that the 1-overall cost of $T_4$, $f(T_4) + rc(T_4, k)$, in any disaster case up to four facility loss (i.e., for any $k < 4$), is strictly lower than the other teams.

## 4 COMPUTATIONAL COMPLEXITY

This section provides a computational complexity analysis of the recoverability issue. More precisely, we consider the following decision problem:

**Definition 4.1.** (Recoverable Team Formation Problem - RTF)

- **Input:** A RTF problem description $RTF = \langle A, S, f, \alpha, h \rangle$ such that $f, h$ are computable in polynomial time, and three non-negative integers $c_1, k, c_2$.
- **Question:** Does there exist an efficient team $T \subseteq A$ such that $T$ is $(c_1, k, c_2)$-recoverable?

We assume that the reader is familiar with the complexity class NP (see [24] for more details). Higher complexity classes are defined using oracles. In particular, $\Sigma^p_2 = \text{NP}^\text{NP}$ corresponds to the class of decision problems that are solved in non-deterministic polynomial time by deterministic Turing machines using an oracle for NP in polynomial time, and $\Sigma^p_2$ is the class of decision problems that are solved in non-deterministic polynomial time by deterministic Turing machines using an oracle for $\Sigma^p_2$ in polynomial time.

We remarked previously that the standard TPT problem, i.e., SETF, is equivalent to the Hitting Set problem [8]. Our strategy to provide a hardness result for our problem of interest RTF is as follows. We (i) introduce an adaptation of the recoverability issue to the more canonical Hitting Set problem; (ii) provide a hardness result for the “recoverable” Hitting Set problem; (iii) provide a polynomial-time reduction from it to our SETF problem.

Let us introduce some notations and vocabulary related to the Hitting Set problem. Let $E$ be a finite set of elements called a universe and $C$ be a collection of subsets of $E$, called a set of constraints based on $E$. A set $T \subseteq E$ is said to be a hitting set for $C$ if for every $U \in C$, $T \cap U = \emptyset$.

Given a universe $E$, a set of constraints $C$ based on $E$, two mappings $f, h : E \mapsto \mathbb{N} \cup \{+\infty\}$, and three non-negative integers $c_1, k, c_2$, we say that a given set $T \subseteq E$ is $(c_1, k, c_2)$-recoverable if $T$ is such that $\sum_{x \in T} f(x) \leq c_1$, then the following conditions are satisfied:

- **Question:** Does there exist a set $T \subseteq E$ such that $T$ is $(c_1, k, c_2)$-recoverable?

We get the following complexity result:

**Proposition 4.3.** $\text{REC-HITTING-SET}$ is $\Sigma^p_3$-hard.

**Proof Sketch.** We intend to prove that $\Sigma^p_3$-hardness holds for REC-HITTING-SET by consider a reduction in polynomial time to REC-HITTING-SET from the validity problem for 3-CNF quantified boolean formulas (QBFs) of the form $\exists x \forall y \exists z. a$ where $X = \{x_1, \ldots, x_{2n}\}$, $Y = \{y_1, \ldots, y_n\}$ and $Z = \{z_1, \ldots, z_n\}$ are three disjoint sets of propositional atoms and $a$ is 3-CNF propositional formula such that $\text{Var}(a) = X \cup Y \cup Z$. Consider such a QBF $\exists x \forall y \exists z. a$, and let us associate with it a universe $E$, a set of constraints $C$ based on $E$, cost functions $f, h : E \mapsto \mathbb{N} \cup \{+\infty\}$ and non-negative integers $c_1, k, c_2$ as follows. Let us define:

- the mappings $a, e$ associating any literal over $X$ with an element of $E$, defined for (possibly negated) literal $x_i$ as $a(x_i) = a_i$ and $e(x_i) = e_i$ if $x_i$ is a positive literal, otherwise $a(x_i) = \overline{a}_i$ and $e(x_i) = \overline{e}_i$;
- the mapping $b$ associating any literal over $Z$ with an element of $E$, defined for (possibly negated) literal $z_i$ as $b(z_i) = b_i$ if $z_i$ is a positive literal, otherwise $b(z_i) = \overline{b}_i$;
- the mapping $\tau$ associating any literal over $Y$ with an integer from $\{1, \ldots, n\}$, defined for (possibly negated) literal $y_i$ as $\tau(y_i) = 2i - 1$ if $y_i$ is a negative literal, otherwise $\tau(y_i) = 2i$.

Additionally, we assume that $\alpha$ is viewed as a set of clauses written as $(l_1, l_2, k)$, where $l_1, l_2, l_k$ are literals from $X \cup Y \cup Z$, and such that the literals $l_1, l_2, l_k$ are ordered in such a way that if $l_i \in Y$ (resp. $l_i \in Z$) then $l_i, l_k \in Y \cup Z$, (resp. $l_k \in Y \cup Z$) and if $l_i \in Z$ (resp. $l_i \in Y$) then $l_1, l_k \in Z$, (resp. $l_k \in Z$). Then a clause $(l_1, l_2, k) \in \alpha$ can be of the form $(x_i, x_j, x_k), (x_i, x_j, y_k), (x_i, x_j, z_k), (x_i, y_j, z_k), (x_i, z_j, z_k), (y_i, y_j, z_k), (z_i, z_j, z_k), (z_i, z_j, z_k)$ (each literal is possibly negated.) The presence of clauses of the form $(y_i, y_j, y_k)$ make the QBF trivially not valid, while clauses $(x_i, y_j, z_k)$ can be removed by unit propagation.

Given the QBF $\exists x \forall y \exists z. a$, the universe $E$, functions $f, h$, set of constraints $C$ based on $E$ and integers $c_1, k, c_2$ are characterized in Algorithm 1. Obviously enough, this algorithm runs in polynomial time in the size of the QBF, its input.

Then the proof takes advantage of the following lemmata:

**Lemma 4.4.** If the deployment candidate set $T$ is a hitting set for $C$ and is such that $\sum_{x \in T} f(x) \leq c_1$, then the following conditions are satisfied:

(i) $d^0$ belongs to $T$;
(ii) $\forall i \in \{1, \ldots, 2n\}$, exactly one element among $(a_i, \overline{a}_i)$ is selected in the deployment candidate set $T$;
(iii) $\forall i \in \{1, \ldots, 2n\}$, $e_i$ (resp. $\overline{e}_i$) is selected in the deployment candidate set $T$ if and only if $a_i$ (resp. $\overline{e}_i$) is also selected;

**Lemma 4.5.** If the deployment candidate set $T$ is a hitting set for $C$ such that $\sum_{x \in T} f(x) \leq c_1$, and if one of the following conditions is not satisfied:

(i) $d^0$ belongs to the removal set $U$;
Algorithm 1: Characterization of $E, f, h, C, c_1, k, c_2$

input: The QBF $3XYY3Z.\alpha$
output: $E, f, h, C, c_1, k, c_2$

begin
  // definition of $E, f$ and $h$
  $E \leftarrow \{a^i, b^i+x, b^i\}$;
  $f(a^i) \leftarrow 0$, $f(b^i+x), f(b^i) \leftarrow +\infty$;
  $h(a^i) \leftarrow +\infty$, $h(b^i+x) \leftarrow n(n+1)$, $h(b^i) \leftarrow 0$;

  for $i \leftarrow 1$ to $n$ do
    $E \leftarrow E \cup \{e_{2i-1}, e_{2i}, e_{2i-1}, e_{2i-1}, e_{2i}, e_{2i}^t\}$;
    $f(e_{2i-1}), f(e_{2i-1}), f(e_{2i}), f(e_{2i}) \leftarrow 1$;
    $h(e_{2i-1}), h(e_{2i-1}), h(e_{2i}), h(e_{2i}) \leftarrow +\infty$;
    $E \leftarrow E \cup \{b_i, b_i, b_i^t\}$,
    $f(b_i), f(b_i), f(b_i^t) \leftarrow +\infty$;
    $h(b_i), h(b_i) \leftarrow 1$, $h(b_i^t) \leftarrow n$;

  // definition of $C$
  $C \leftarrow \{(a^i, b^i)_i\}$;

  for $i \leftarrow 1$ to $n$ do
    $C \leftarrow C \cup \{(e_{2i-1}, b_{2i}^t), (b_{2i}, e_{2i}, b_{2i})\}$;
    $C \leftarrow C \cup \{(e_{2i}, e_{2i-1}, b_{2i}^t), (e_{2i}, e_{2i}, b_{2i})\}$;
    $C \leftarrow C \cup \{(e_{2i}, b_{2i}, e_{2i-1}, b_{2i}), (a^i, b_{2i}, b_{2i})\}$;

  for $i \leftarrow 1$ to $n$ do
    $C \leftarrow C \cup \{(a^i, b_i, b_i^t)\}$;

  for $(l_i, l_p, l_k)$ in $C$ do
    switch $(l_i, l_p, l_k)$ do
      case $[x_i, y_i, z_i]$: do
        $C \leftarrow C \cup \{a(x_i), a(y_i), a(z_i), b^i\}$;
        $C \leftarrow C \cup \{a(x_i), a(y_i), b^i\}$;
      case $[a(z_i), b(z_i)]$: do
        $C \leftarrow C \cup \{a^i, b^i, b(z_i), b(z_i)\}$;
      case $[a(y_i), b^i, b(z_i)]$: do
        $C \leftarrow C \cup \{a^i, a(y_i), a(z_i), b^i, b(z_i)\}$;
      case $[a(y_i), a(z_i), b(z_i)]$: do
        $C \leftarrow C \cup \{a^i, a(y_i), a(z_i), a(z_i), b(z_i)\}$;
      case $[a(x_i), y_i, z_i]$ do
        $C \leftarrow C \cup \{a^i, e(x_i), e(x_i), b(z_i)\}$;
      case $[a(x_i), z_i, b(z_i)]$: do
        $C \leftarrow C \cup \{a^i, e(x_i), b(z_i), b(z_i)\}$;
      case $[a(x_i), y_i, z_i]$ do
        $C \leftarrow C \cup \{a^i, b(x_i), e(x_i), b(z_i)\}$;
    // definition of integers $c_1, k, c_2$
    $c_1 \leftarrow 4n$, $k \leftarrow n + 1$, $c_2 \leftarrow 2n(n+1)$.

\[ \sum_{i \in A \cap \text{net}(i)} x_i \geq 1 \quad \forall s \in S \text{ (Def.2.3) } \]
\[ x_i \text{ does not violate any cuts } \text{(Eq.4) } \]
\[ x_i \in \{0, 1\} \quad \forall i \in A \text{ (3) } \]

Taking advantage of Prop. 4.3, we are now ready to characterize the complexity of RTF. Indeed, REC-HITTING-SET and RTF are equivalent problems:

**Proposition 4.7.** RTF is $\Sigma^P_3$-complete.

Intuitively, the increase in complexity stems from the fact that for each feasible team $T$, one needs to consider every possible removal of $k$ agents and compute its $k$-recovery cost. The number of combinations is exponential with respect to $k$, and for each removal computing the $k$-recovery cost amounts to solving a standard TF problem, which is itself NP-hard. In a sense, the parameter $k$ is the reason for the complexity shift. If we consider $k$ as a constant that is part of the problem definition (denoted as $k$-RTF), rather than as an input parameter, we obtain a significant drop in computational complexity:

**Proposition 4.8.** For $k \geq 0$, $k$-RTF is NP-complete.

## 5 ALGORITHMS

We now describe our complete and heuristic algorithms. Given an RTF description and an integer $k$, our goal is to find a team $T$ that minimizes the $k$-overall cost (Def. 3.2). We call such an optimal team, denoted $T_{\text{best}}$, a $k$-optimal team. An outline of our procedure is given in Algorithm 2. Iteratively, we compute a team that satisfies the constraints of the problem. Based on the team, we generate a cut (Eq. 4), which is added to the set of constraints, that prunes suboptimal teams with a similar substructure from the search space. Therefore, at the end of the algorithm, a $k$-optimal team $T_{\text{best}}$ is produced.

The cuts are nonlinear. However, by traversing the search space in increasing cost of the teams, they can be simplified. To further speed up the algorithm at the cost of losing optimality, we introduce heuristic cuts. The main components of the algorithm are discussed in the following sections.

### 5.1 Computing Teams

We model TF as a set of linear equations with binary variables. Initially, the set of equations consists of the efficiency constraints (Eq. 1). In each iteration, a new team is computed that satisfies all the equations. Based on the newly computed team, a new equation (cut) is added to the set of equations (Eq. 2), which prunes suboptimal teams (wrt the $k$-overall cost) with similar substructure from future search. This process is repeated until no new teams satisfying the equations exist. At this point, the complete search space has been explored and the optimal solution has been found in one of the prior iterations. In our model, a solution $x = (x_1, x_2, \ldots, x_{|A|})$ is the assignment of values to the binary decision variables $x_i$, which indicate if the $i$th agent is in the team $T$. Thus, every assignment for the $x_i$ variables uniquely corresponds to a team and vice versa. The model used is as follows:

\[ \sum_{i \in A \cap \text{net}(i)} x_i \geq 1 \quad \forall s \in S \text{ (Def.2.3) } \]
\[ x_i \text{ does not violate any cuts } \text{(Eq.4) } \]
\[ x_i \in \{0, 1\} \quad \forall i \in A \text{ (3) } \]
In our experiments, solutions were obtained by using IBM ILOG CPLEX 12.8.0, an integer programming (IP) solver. The IP solver is a search-based branch-and-bound algorithm, which is complete: it is guaranteed to find a solution, if it exists. We note that solving TF is a NP-hard problem and hence in each iteration we make a call to a NP-hard solver.

Algorithm 2: Algorithm Outline for RTF

input: An RTF description \((A, S, f, \alpha, h)\), an integer \(k\)
output: A k-optimal team \(T_{best}\)

1 begin
2 \(T_{best} \leftarrow \emptyset\)
3 while not all feasible teams have been explored do
4 \(T_{cur} \leftarrow \text{selectNewTeam}()\)
5 \(\text{generateCut}(T_{cur}, A_k, T_{best},RTF,k)\)
6 if \(oc(T_{cur}, k) < oc(T_{best}, k)\) then
7 \(T_{best} \leftarrow T_{cur}\)
8 return \(T_{best}\)

5.2 Cut for Pruning Suboptimal Teams

Given a solution \(T_{cur}\) to Eq.1-3, the goal is to produce a cut (an equation) that prunes suboptimal teams similarly structured as \(T_{cur}\).

Before providing the cut, we introduce the following notation:

\(M(S)\) is the set of agents that posses at least one of the skills from the set of skills \(S\), \(S(T, A_k)\) is the set of skills that become uncovered when agents in \(A_k\) are removed from \(T\). \(T_{best}\) is the k-optimal team found so far during the search, and \(C^{\text{thresh}}(T, T_{best}) = oc(T_{best},k) - f(T) - 1\) is the threshold value of the k-recovery cost for team \(T\) that must not be exceeded to be considered better than \(T_{best}\). Note that computing \(rc(T, A_k)\) (Def. 3.2) amounts to solving a team formation problem with the skills \(S(T, A_k)\) and agents \(M(S(T, A_k))\) \(\cap\ T\).

Let \(T_{cur}\) be the currently analyzed team, \(A_k\) is the set of agents such that \(rc(T_{cur}, A_k) > C^{\text{thresh}}(T_{cur}, T_{best})\), and \(T\) is any future team uniquely defined by the variables \(x_i\) that would be considered as a solution to Eq.1-3. We now present the cut:

\[
C^{\text{thresh}}(T, T_{best}) < rc(T_{cur}, A_k) \Rightarrow \sum_{i \in M(S(T_{cur}, A_k)) \cap A_k} x_i \geq 1 \quad (4)
\]

The cut forces the inclusion of at least one agent from the set \(M(S(T, A_k)) \cap A_k\) under the condition that the left-hand side is satisfied. If the cut is violated, the resulting k-recovery cost will be greater than the threshold value. Essentially, the partial assignment of \(T_{cur}\) concerning agents from \(M(S(T, A_k)) \cap A_k\) is responsible for the high k-recovery cost.

Example 5.1. Consider a cut for the 1-recoverable health care facility location example provided in the introduction. To simplify, we only allow building clinics in the first step and rescue centers in the second step. For the team \(T_{cur}\) given in Fig. 2(a), the set \(A_k\) consists of a single clinic (circled), \(f(T_{cur}) = 80\), \(rc(T_{cur}, A_k) = 40\), and we assume that \(T_{cur}\) is the best team found so far (\(T_{best} = T_{cur}\)). The area that would become uncovered after the removal of \(A_k\) (\(S(T_{cur}, A_k)\)) is marked in Fig. 2(b). Fig. 2(c) shows the possible clinics that can cover the uncovered area \((M(S(T_{cur}, A_k)))\). In Fig. 2(d) the same clinics without without \(A_k\) are shown. The cut is then \(119 - f(T) < 40 \Rightarrow x_{1,2} + x_{2,2} + x_{1,3} + x_{3,3} \geq 1\), which states that for every team \(T\) with \(f(T) \geq 80\), at least one of the clinics from the right-hand side (Fig. 2(d)) must be built, otherwise \(T\) will have a 1-recovery cost of at least 40, meaning it would not be better than \(T_{best}\). One of the teams pruned from the search space by the generated cut is given in Fig. 2(e).

Proposition 5.2. The cut in Eq. 4, given two teams \(T_{best}\) and \(T_{cur}\) with \(oc(T_{best}, k) \leq oc(T_{cur}, k)\), and \(A_k\) such that \(rc(T_{cur}, A_k) > C^{\text{thresh}}(T_{cur}, T_{best})\), does not prune any team \(T\) for which \(oc(T, k) < oc(T_{best}, k)\) holds.

Given that both the integer programming search procedure to calculate teams is complete and cuts (Eq. 4) does not exclude optimal solutions (Prop. 5.2), we arrive at the following corollary:

Corollary 5.3. Algorithm 2 is complete, that is, it computes the optimal solution.

The final issue remaining is computing \(A_k\). This is done by enumeration, searching for the \(A_k\) that yields the maximum value \(\epsilon_{rec} = rc(T_{cur}, A_k)\) or stopping at the first \(A_k\) that shows that \(T_{cur}\) is suboptimal, as outlined in Algorithm 3.

Algorithm 3: Cut Generation for Pruning Teams

input: RTF = \(\{A, S, f, \alpha, h\}\), \(k\), a pruning team \(T_{cur}\), best team \(T_{best}\)
output: a cut pruning teams with same substructure as input team \(T_{cur}\)

1 begin
2 \(\epsilon_{max} \leftarrow 0\)
3 \(A_k^* \leftarrow \emptyset\)
4 // calculate the maximum k-recovery cost w.r.t. // removals of \(A_k\)
5 foreach \(A' \in \{A' : A' \subseteq T_{cur} \land |A'| = k\}\) do
6 \(\epsilon_{rec} = h(T_{cur}, A_k)\)
7 if \(\epsilon_{rec} \geq \epsilon_{max}\) then
8 \(\epsilon_{max} \leftarrow \epsilon_{rec}\)
9 \(A_k^* \leftarrow A_k\)
10 if \(C^{\text{thresh}}(T_{cur}, T_{best}) < \epsilon_{max}\) then
11 exit loop
12 return Eq. 4 and 5 w.r.t. \((T_{cur}, A_k^*, T_{best}, RTF)\)

5.3 Search Strategy for Cut Simplification

A key observation is that if we knew the deployment cost of the team we are searching for, the left-hand side of the cut in Eq. 4 would be obsolete. We exploit this information in our search strategy: we traverse the search space by considering teams in increasing deployment cost. Therefore, teams \(T\) are examined before teams \(T'\) when \(f(T) < f(T')\). This effectively linearizes the cut, as the left-hand side is no longer necessary. Note that every cut generated based on a team \(T_1\) is still valid even for teams \(T_2\) with \(f(T_1) \leq f(T_2)\). Such linear cuts are preferred over nonlinear cuts, as they are typically easier to solve in practice.
We considered small, medium, and large random instances. The comparison with robustness is shown in Fig. 12.8.0 and integer programming in C++, as it was significantly faster than the base cut. The parameter $k$ and the gap increases as $k$ grows; which confirms our expectations given our running example. Therefore, for applications where recoverability is sufficient, more desirable results can be achieved with recoverability than with robustness. Robustness provides desirable properties and is easier to compute, but it is noticeably more expensive. The parameter $k$ plays an important role in the computational time, as expected due to Prop. 4.8. The heuristic cut (Eq. 5) provides a means to combat the increase in computational time.

## 6 EXPERIMENTAL RESULTS

The goal of our experimentation is to show that recoverability has a much lower cost than robustness, at the expense of higher computational complexity. Solutions for reasonably sized teams can be found, despite RTF being a $\Sigma P^2$-hard problem. The results are presented in Fig. 3 and Tables 2 and 3.

### 6.1 Experimental Setup

We considered small, medium, and large random instances. The small instances (from 10 to 30 agents and 3 to 11 skills) correspond to the robust TF instances given in [23]. The medium ones (from 100 to 150 agents and 20 to 30 skills) are set covering instances used in [2, 29]. The large ones (1000 agents and 200 skills) are classical instances found in the OR Library [1] under the scipx package, used in set covering works (e.g. [7, 15, 20]). In our experiments we set $h = f$ (recall Definition 3.3).

All tests were performed on an Intel Core i7-7700HQ CPU @ 2.80GHz with 32 GB RAM using a computation limit of one hour. To compute solutions for robust TF, we used IBM ILOG CPLEX 12.8.0 and integer programming in C++, as it was significantly faster than the previous approaches proposed in [5] and [23].

### 6.2 Comparison with Robust Team Formation

The comparison with robustness is shown in Fig. 3 for instances with 150 agents and 30 skills (the values are averaged over ten instances). Small instances from [23] are solved within seconds for $k = \{1, 2, 3\}$. The $k$-optimal teams $T$ in RTF have a significantly lower $k$-overall cost than $k$-robust TF for the different values of $k$, and the gap increases as $k$ grows; which confirms our expectations given our running example. Therefore, for applications where recoverability is sufficient, more desirable results can be achieved with recoverability than with robustness. Robustness provides desirable properties and is easier to compute, but it is noticeably more expensive. The parameter $k$ plays an important role in the computational time, as expected due to Prop. 4.8. The heuristic cut (Eq. 5) provides a means to combat the increase in computational time.

### 6.3 Algorithm Variants

We considered adding the heuristic cut (Eq. 5) in addition to the base cut (Eq. 4), and our search strategy of going through teams in increasing deployment cost. Detailed results in Table 2 on a few selected instances (as similar results are obtained for other instances). Aggregated results for small and medium benchmarks are given in Table 3. Overall, both our search strategy and our heuristic cut have more impact as the size of the instance and $k$ grow, reducing the execution time while maintaining high solution quality.

## 7 RELATED WORK

The notions of recoverability and robustness have been introduced as general concepts for resilient systems. Robustness can be seen as one of its two main characteristics [4], as the capability to keep some “level of demand” without suffering degradation, while recoverability is defined as the capacity to cope with unanticipated dangers after they have become effectively manifested on the system. Recoverability has also been considered for constraint-based dynamic systems in [26], where the authors reviewed a number...
of qualitative definitions of resilience in various disciplines and formalized the property of resilience as a combination of several properties. Among them, resistance and recoverability were considered. Resistance deals with the system’s ability to resist to external fluctuations: a system’s trajectory is l-resistant if its “quality” is always kept above a certain threshold l. Recoverability refers to the system’s “elasticity”, i.e., its ability return to a satisfactory state after temporarily suffering from some external disturbances: a system’s trajectory is (p, q)-recoverable if whenever its “quality” goes below a certain threshold p, it bounces back to an acceptable state (above p) within a certain window of accumulative cost specified by q. The latter definition (see [26] for more formal details) is based on the same intuition as in this paper in the context of TF. Interestingly, it can be checked that l-resistance coincides with (p, q)-recoverability in the case where q = 0, i.e., when no accumulative cost is allowed at all; this in accordance with our results (Prop. 3.4), which shows that in TF, recoverability can be viewed as a generalization of robustness.

TF is somewhat related to coalition structure generation (CSG), where the task is to partition a set of agents into teams (so-called “coalitions”) in order to maximize the sum of the coalition utility values. However, while in TF a single team is formed out of a pool of agents, in CSG already preselected agents are split into teams. Thus TF and CSG are respectively related to set covering and set partitioning so that our results cannot be trivially adapted. We direct the interested reader to [25] for a survey on CSG.

Robust TF was extended in [22] to handle multiple teams. Heuristic and approximation algorithms for robust TF were studied in [5].

On the algorithmic side, the cut component of our algorithm for RTF is related to clause learning for Boolean satisfiability (SAT) [27, 28] and no goods in constraint programming (CP) [6, 21], which also analyze conflicting solutions and generate cuts. However, these techniques are not applicable to RTF, since they are designed for NP-complete problems, while RTF is $\Sigma^P_3$-complete.

The notion of resilience is related to previous work on robustness for combinatorial problems. The notion of (a, b)-super models was introduced [10] for Boolean satisfiability. A (a, b)-super model is a propositional model such that if variables in a set of size a or less change their values (breakage), another model can be obtained by modifying the value of a disjoint set of variables of size at most b (repair). The notion was later extended to (a, b)-super solutions [12] for constraint satisfaction problems, where the focus is laid on the special case of (1, 0)-super solutions. Optimization variants were also studied, such as computing the (1, 0)-super solution that maximizes the number of repairable variables [12], or finding the best model that is a (1, 0)-super solution [11].

For certain problems, basing robustness solely on the number of repairs is not appropriate, as some repairs might be easier to perform than others. Such a formalism, which is closest to our work, is the extension for maxSAT [3], where the task is to compute a solution with minimum cost such that after changing the values of any a variables, it can be repaired by changing the values of other b variables, so that the cost does not decrease by more than a given $\beta$. Probabilistic weighted variants, named (a, $\beta$)-weighted super solutions, were introduced [13], where a is substituted by a, the breakage probability, and $\beta$, the cost of the repair.

While the mentioned super solution works share similarities, there are notable differences with our work. First, the constraint programming based techniques [12] cannot be practically adapted to our problem. Indeed, constraint programming approaches are known to be ineffective for set covering (and thus, for TF) when compared to integer programming approaches. This is why we introduced here a novel cut-based algorithm for solving RTF. Second, the reformulation approach for maxSAT [3] adds an exponential number of constraints ($O(n^a)$ with n being the number of variables) and thus seems unpromising for RTF. Last but not least, to the best of our knowledge, the computational complexity of the problem (cf. Prop. 4.7) was yet unknown. In contrast, the complexity of the counterpart problem in computing super models and super solutions has been characterized only for the special case when the parameter a is a constant, which is analogous to Prop. 4.8.

### 8 CONCLUSION

We introduced a framework for recoverable team formation (RTF), where in addition to the team’s cost we examine the cost related to restoring its functionality after $k$ agents have been removed. Therefore, we provided a framework for building teams resilient to change, which is more general than the work in [23], at the expense of a complexity shift to $\Sigma^P_3$-hardness. Our algorithms, by using our proposed cuts and search strategy, were able to solve reasonably sized problems despite the high computational complexity. Our results have shown the drastic difference in overall cost, in favor of recoverability. Computing multiple teams, stochastic settings, and introducing recoverability to other computational problems are all topics for future work.
REFERENCES


