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

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

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

Katsumi Inoue
National Institute of Informatics, Tokyo, Japan
Tokyo Institute of Technology, 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:
Emir Demirović, Nicolas Schwind, Tenda Okimoto, and Katsumi Inoue. 2018. Recoverable Team Formation: Building Teams Resilient to Change. In *Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), Stockholm, Sweden, July 10–15, 2018,* IFAAMAS, 9 pages.

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.

Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018), M. Dastani, G. Sukthankar, E. André, S. Koenig (eds.), July 10–15, 2018, Stockholm, Sweden. © 2018 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

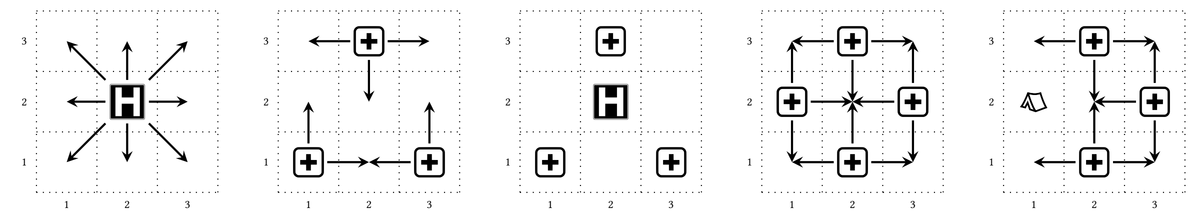

(a) An efficient team T_1 (deployment cost: 80).
(b) An optimal efficient team T_2 (deployment cost: 60).
(c) An optimal 1-robust team T_3 (deployment cost: 140).
(d) An optimal 1-recoverable team T_4 (deployment cost: 80).
(e) Recovering T_4 after a clinic collapses (1-recovery cost: 20).

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

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) iff $|x - x_F| + |y - y_F| \leq r_F$. 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 rending 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 $\Sigma_3^{\mathbf{P}}$-complete. Given that the standard and robust TF problems are both "only" **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 $TF = \langle A, S, f, \alpha \rangle$ 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, $f : 2^A \rightarrow \mathbb{N}$ is a cost function, and α 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 satisfies $f(T) = \sum_{a_i \in T} f(\{a_i\})$.

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

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

[2]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 $A = \{a_1, a_2, ..., a_n\}$ is a set of agents, $S = \{s_1, s_2, ..., s_m\}$ is a set of skills, $f : 2^A \rightarrow \mathbb{N} \cup \{+\infty\}$ is the deployment cost function, α 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

$$rc(T, k) = \max_{\substack{T' \subseteq T, \\ |T'|=k}} rcS(T, T'),$$

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

$$rcS(T, T') = \min_{\substack{T_{rec} \subseteq A \setminus T, \\ (T \setminus T') \cup T_{rec} \text{ efficient}}} h(T_{rec}).$$

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 $\langle k, c_2\rangle$*-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 $\langle c_1, k, c_2\rangle$-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 $\langle c, k, 0\rangle$-recoverable.*

Example 3.5 (continued). Let us formalize the health care facility location example provided in the introduction. For each facility of type $t = \{H, C, R\}$ (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^t_{xy}. 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 $A = \{a^t_{xy} \mid t \in \{C, R\}, x, y \in \{1, 2, 3\}\} \cup \{a^H_{22}\}$. 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 $a^t_{xy} \in A$ 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 $a^t_{xy} \in A$, $\alpha(a^t_{xy}) = \{s_{ij} \in S \mid |x - i| + |y - j| \leq range(t)\}$, 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 introductive example (cf. Fig. 1) are defined as $T_1 = \{a^H_{12}\}$, $T_2 = \{a^C_{11}, a^C_{31}, a^C_{23}\}$, $T_3 = T_1 \cup T_2$, and $T_4 = \{a^C_{12}, a^C_{21}, a^C_{23}, a^C_{32}\}$. 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_2 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, $\langle 80, 1, 20\rangle$-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

T_i	$f(T_i)$	$rc(T_i,1)$	$rc(T_i,2)$	$rc(T_i,3)$	$rc(T_i,4)$
T_1	80	180	180	180	180
T_2	60	80	120	180	180
T_3	140	0	80	120	180
T_4	80	20	60	100	180

Table 1: The (k-recovery) costs of each team.

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 $\langle c_1, k, c_2\rangle$-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, $\Sigma_2^{\mathbf{P}} = \mathbf{NP}^{\mathbf{NP}}$ 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 $\Sigma_3^{\mathbf{P}}$ is the class of decision problems that are solved in non-deterministic polynomial time by deterministic Turing machines using an oracle for $\Sigma_2^{\mathbf{P}}$ in polynomial time.

We remarked previously that the standard TF 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 $U \in C, T \cap U \neq \emptyset$.

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 $\langle c_1, k, c_2\rangle$-recoverable if T is such that $\sum_{x \in T} f(x) \leq c_1$, T is a hitting set for C, and for any set $U \subseteq T$ with $|U| \leq k$, there exists a set $V \subseteq E \setminus T$ such that $\sum_{x \in V} h(V) \leq c_2$ and $(T \setminus U) \cup V$ is a hitting set for C.

*Definition 4.2. (*REC-HITTING-SET*)*

- **Input:** A universe E, a set of constraints C based on E, two functions $f, h : E \mapsto \mathbb{N} \cup \{+\infty\}$ computable in polynomial time, and three non-negative integers c_1, k, c_2.
- **Question:** Does there exist a set $T \subseteq E$ such that T is $\langle c_1, k, c_2\rangle$-recoverable?

We get the following complexity result:

Proposition 4.3. REC-HITTING-SET is $\Sigma_3^{\mathbf{P}}$-hard.

Proof sketch. We intend to prove that $\Sigma_3^{\mathbf{P}}$-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 $\exists X \forall Y \exists Z.\alpha$ 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 α is 3-CNF propositional formula such that $Var(\alpha) = X \cup Y \cup Z$. Consider such a QBF $\exists X \forall Y \exists Z.\alpha$, 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 every (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 $a(x_i) = \bar{a}_i$ and $e(x_i) = \bar{e}_i$;
- the mapping b associating any literal over Z with an element of E, defined for every (possibly negated) literal z_i as $b(z_i) = b_i$ if z_i is a positive literal, otherwise $b(z_i) = \bar{b_i}$;
- the mapping τ associating any literal over Y with an integer from $\{1, \ldots, n\}$, defined for every (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 α is viewed as a set of clauses written as (l_i, l_j, l_k), where l_i, l_j, l_k are literals from $X \cup Y \cup Z$, and such that the literals l_i, l_j, l_k are ordered in such a way that if $l_i \in Y$ (resp. $l_j \in Y$) then $l_j, l_k \in Y \cup Z$, (resp. $l_k \in Y \cup Z$) and if $l_i \in Z$ (resp. $l_j \in Z$) then $l_j, l_k \in Z$, (resp. $l_k \in Z$). Then a clause $(l_i, l_j, l_k) \in \alpha$ 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), (y_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, y_k) can be removed by unit propagation.

Given the QBF $\exists X \forall Y \exists Z.\alpha$, 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) *a^0 belongs to T;*
(ii) *$\forall i \in \{1, \ldots, 2n\}$, exactly one element among $\{a_i, \bar{a}_i\}$ is selected in the deployment candidate set T;*
(iii) *$\forall i \in \{1, \ldots, 2n\}$, e_i (resp. $\bar{e}_i$) is selected in the deployment candidate set T if and only if a_i (resp. $\bar{e}_i$) 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) *a^0 belongs to the removal set U;*

Algorithm 1: Characterization of E, f, h, C, c_1, k, c_2

```
input: The QBF $\exists X \forall Y \exists Z . \alpha$
ouput: $E, f, h, C, c_1, k, c_2$
begin
    // definition of $E, f$ and $h$
    $E \leftarrow \{a^0, b^{++}, b^0\}$;
    $f(a^0) \leftarrow 0$;    $f(b^{++}), f(b^0) \leftarrow +\infty$;
    $h(a^0) \leftarrow +\infty$;    $h(b^{++}) \leftarrow n(n+1)$;    $h(b^0) \leftarrow 0$;
    for $i \leftarrow 1$ to $n$ do
        $E \leftarrow E \cup \{a_{2i-1}, \bar{a}_{2i-1}, a_{2i}, \bar{a}_{2i}, e_{2i-1}, \bar{e}_{2i-1}, e_{2i}, \bar{e}_{2i}\}$;
        $f(a_{2i-1}), f(\bar{a}_{2i-1}), f(a_{2i}), f(\bar{a}_{2i}) \leftarrow 1$;
        $f(e_{2i-1}), f(\bar{e}_{2i-1}), f(e_{2i}), f(\bar{e}_{2i}) \leftarrow 1$;
        $h(a_{2i-1}), h(\bar{a}_{2i-1}), h(a_{2i}), h(\bar{a}_{2i}) \leftarrow +\infty$;
        $h(e_{2i-1}), h(\bar{e}_{2i-1}), h(e_{2i}), h(\bar{e}_{2i}) \leftarrow +\infty$;
        $E \leftarrow E \cup \{b_i, \bar{b}_i, b_i^+\}$;
        $f(b_i), f(\bar{b}_i), f(b_i^+) \leftarrow +\infty$;
        $h(b_i), h(\bar{b}_i) \leftarrow 1$;    $h(b_i^+) \leftarrow n$;
    // definition of $C$
    $C \leftarrow \{\{a^0, b^{++}\}\}$;
    for $i \leftarrow 1$ to $n$ do
        $C \leftarrow C \cup \{\{a_{2i-1}, \bar{a}_{2i-1}, b_i^+\}, \{a_{2i}, \bar{a}_{2i}, b_i^+\}\}$;
        $C \leftarrow C \cup \{\{e_{2i-1}, \bar{e}_{2i-1}, b^0\}, \{e_{2i}, \bar{e}_{2i}, b^0\}\}$;
        $C \leftarrow C \cup \{\{a_{2i-1}, \bar{e}_{2i-1}, b^0\}, \{a_{2i}, \bar{e}_{2i}, b^0\}\}$;
    for $i \leftarrow 1$ to $m$ do
        $C \leftarrow C \cup \{\{a^0, b_i, \bar{b}_i\}\}$;
    for $(l_i, l_j, l_k)$ in $\alpha$ do
        switch $(l_i, l_j, l_k)$ do
            case $(x_i, x_j, x_k)$ do
                $C \leftarrow C \cup \{\{a(x_i), a(x_j), a(x_k), b^0\}\}$;
            case $(x_i, x_j, y_k)$ do
                $C \leftarrow C \cup \{\{a(x_i), a(x_j), b^0\}\}$;
            case $(z_i, z_j, z_k)$ do
                $C \leftarrow C \cup \{\{a^0, b(z_i), b(z_j), b(z_k)\}\}$;
            case $(y_i, z_j, z_k)$ do
                $C \leftarrow C \cup \{\{a_{\tau(y_i)}, \bar{a}_{\tau(y_i)}, b(z_j), b(z_k)\}\}$;
            case $(y_i, y_j, z_k)$ do
                $C \leftarrow C \cup \{\{a_{\tau(y_i)}, \bar{a}_{\tau(y_i)},$
                        $a_{\tau(y_j)}, \bar{a}_{\tau(y_j)}, b(z_k)\}\}$;
            case $(x_i, x_j, z_k)$ do
                $C \leftarrow C \cup \{\{a^0, e(x_i), e(x_j), b(z_k)\}\}$;
            case $(x_i, z_j, z_k)$ do
                $C \leftarrow C \cup \{\{a^0, e(x_i), b(z_j), b(z_k)\}\}$;
            case $(x_i, y_j, z_k)$ do
                $C \leftarrow C \cup \{\{a^0, e(x_i), a_{\tau(y_j)}, \bar{a}_{\tau(y_j)}, b(z_k)\}\}$;
    // definition of integers $c_1, k, c_2$
    $c_1 \leftarrow 4n$;    $k \leftarrow n + 1$;    $c_2 \leftarrow 2n(n+1)$;
```

(ii) $\forall i \in \{1, \ldots, n\}$, *exactly one element from* $\{a_{2i-1}, \bar{a}_{2i-1}, a_{2i}, \bar{a}_{2i}\}$ *belongs to the set* U*;*

(iii) $\forall i \in \{1, \ldots, n\}$, *no element from* $\{e_{2i-1}, \bar{e}_{2i-1}, e_{2i}, \bar{e}_{2i}\}$ *belongs to the set* U*,*

then there exists a k-recovery candidate set $V \subseteq E \setminus T$ *such that* $\sum_{x \in V} h(V) \leq c_2$ *and* $(T \setminus U) \cup V$ *is a hitting set for* C.

Lemma 4.6. *If the deployment candidate set* T *is a hitting set for* C *such that* $\sum_{x \in T} f(x) \leq c_1$*, all conditions (i), (ii) and (iii) from Lemma 4.5 are satisfied, and there exists a set* $T \subseteq E$ *that is* $\langle c_1, k, c_2 \rangle$*-recoverable, then the k-recovery candidate set* V *is such that for each* $i \in \{1, \ldots, n\}$*, exactly one element among* $\{b_i, \bar{b}_i\}$ *belongs to* V.

Then by using lematta 4.4, 4.5 and 4.6 above, one can verify that the QBF is valid if and only if there exists a set $T \subseteq E$ which is $\langle c_1, k, c_2 \rangle$-recoverable. Therefore, REC-HITTING-SET is $\Sigma_3^{\mathbf{P}}$-hard. □

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 $\Sigma_3^{\mathbf{P}}$-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_{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_{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 $x = (x_1, x_2, ..., x_{|A|})$ 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:

$$\sum_{i \in A \wedge s \in \alpha(i)} x_i \geq 1 \qquad \forall s \in S \text{ (Def.2.3)} \tag{1}$$

$$x \text{ does not violate any cuts (Eq.4)} \tag{2}$$

$$x_i \in \{0, 1\} \qquad \forall i \in A \tag{3}$$

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 $\langle A, S, f, \alpha, h \rangle$, an integer $k$
    ouput: A $k$-optimal team $T_{best}$
1   begin
2       $T_{best} \longleftarrow \emptyset$
3       while $not\ all\ feasible\ teams\ have\ been\ explored$ do
4           $T_{cur} \longleftarrow selectNewTeam()$
5           $generateCut(T_{cur}, a^*_k, T_{best}, RTF, k)$
6           if $oc(T_{cur}, k) < oc(T_{best}, k)$ then
7               $T_{best} \longleftarrow T_{cur}$
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 $C^{thresh}(T, T_{best}) = oc(T_{best}, k) - f(T) - 1$ 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 $M(S(T, A_k)) \setminus T$.

Let T_{cur} be the currently analyzed team, A^*_k is the set of agents such that $rc(T_{cur}, A^*_k) > C^{thresh}(T_{cur}, T_{best})$, 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:

$$C^{thresh}(T, T_{best}) < rc(T_{cur}, A^*_k) \Rightarrow \sum_{i \in M(S(T_{cur}, A^*_k)) \setminus A^*_k} x_i \geq 1 \qquad (4)$$

The cut forces the inclusion of at least one agent from the set $M(S(T, A^*_k)) \setminus A^*_k$ 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 $M(S(T, A^*_k)) \setminus A^*_k$ 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 further 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^*_1 consists of a single clinic (circled), $f(T_{cur}) = 80$, $rc(T_{cur}, A^*_1) = 40$, 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^*_1 ($S(T_{cur}, A^*_1)$) is marked in Fig. 2(b). Fig. 2(c) shows the possible clinics that can cover the uncovered area ($M(S(T_{cur}, A^*_1))$). In Fig. 2(d) the same clinics without without A^*_1 are shown. The cut is then $119 - f(T) < 40 \Rightarrow x_{(1,2)} + x_{(2,2)} + x_{(2,3)} + x_{(3,3)} \geq 1$, 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 $rc(T_{cur}, A^*_k) > C^{thresh}(T_{cur}, T_{best})$, 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.*

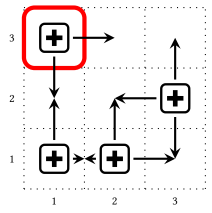
The final issue remaining is computing A^*_k. This is done by enumeration, searching for the A^*_k that yields the maximum value $c_{rec} = rc(T_{cur}, A^*_k)$ or stopping at the first A^*_k that shows that T_{cur} is suboptimal, as outlined in Algorithm 3.

Algorithm 3: Cut Generation for Pruning Teams

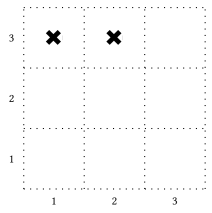
```
    input: $RTF = \{A, S, f, \alpha, h\}$, $k$, a pruning team $T_{cur}$, best
           team $T_{best}$
    ouput: a cut pruning teams with same substructure as input
           team $T_{cur}$
1   begin
2       $c^{max}_{rec} \longleftarrow 0$
3       $A^*_k \longleftarrow \emptyset$
        // calculate the maximum k-recovery cost w.r.t.
        // removals of $A_k$
4       foreach $A_k \in \{A' : A' \subseteq T_{cur} \land |A'| = k\}$ do
5           $c_{rec} = h(T_{cur}, A_k)$
6           if $c_{rec} \geq c^{max}_{rec}$ then
7               $c^{max}_{rec} \longleftarrow c_{rec}$
8               $A^*_k \longleftarrow A_k$
9               if $C^{thresh}(T_{cur}, T_{best}) < c^{max}_{rec}$ then
10                  $exit\ loop$
11      return $Eq.$ 4 and 5 w.r.t. $(T_{cur}, A^*_k, T_{best}, RTF)$
```

5.3 Search Strategy for Cut Simplification

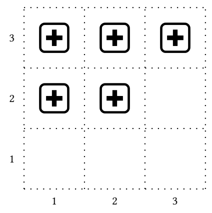
A key observation is that if we knew the deployment cost of the team we are searching for, the left-hand side of the cut in Eq. 4 would be obsolete. We exploit this information in our search strategy: we traverse the search space by considering teams in increasing deployment cost. Therefore, teams T 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.



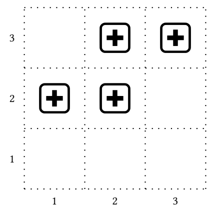
(a) An efficient team T_{cur} (deployment cost: 80 and 1-recovery: 40) with singleton A_1^* circled.



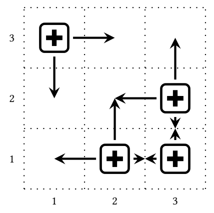
(b) Areas that become uncovered after A_1^* is removed from T_{cur} ($S(T_{cur}, A_1^*)$)



(c) Possible clinics that can cover the uncovered areas ($M(S(T_{cur}, A_1^*))$).



(d) Possible clinics without A_1^* ($M(S(T_{cur}, A_1^*)) \setminus A_1^*$)



(e) A team pruned by the cut generated based on T_{cur}.

Figure 2: Illustrative example: cut generation.

5.4 Heuristic Cut

We introduce a heuristic cut as a means of reducing the computational time at the expense of possibly losing optimality. In the base cut (Eq. 4), the removal agents A_k^* are the cause of high k-recovery costs. The intuition for the heuristic cut is that by pruning all teams with A_k^* the same problem cannot be encountered again:

$$\sum_{i \in A_k^*} x_i \leq |A_k^*| - 1 \tag{5}$$

The cut has stronger pruning power, but it might remove the optimal solution, therefore the algorithm becomes incomplete. In our experimental analysis, we observed that good solutions can still be obtained while using the heuristic cut. In our heuristic algorithm variant, the heuristic cut is generated in addition to the base cut.

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 $\Sigma_3^{\mathbf{P}}$-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 *scp4x* 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

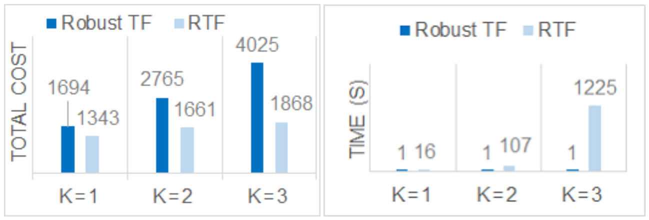


Figure 3: Comparison on instances with 150 agents and 30 skills. Average for ten instances presented.

$k = \{1, 2, 3\}$. 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

$(\lvert A\rvert,\lvert S\rvert)$	**k**	**(s,~~h~~)**	**(s,h)**	**(~~s~~,~~h~~)**	**(~~s~~,h)**
(150, 30)	1	1478 (26s)	1536 (20s)	1478 (5s)	1536 (1s)
(150, 30)	2	1828 (260s)	1828 (60s)	1828 (350s)	1828 (9s)
(150, 30)	3	1962 (1820s)	1962 (160s)	1962 (2640s)	1962 (270s)
(1000, 200)	1	487 (45s)	487 (20s)	487 (20s)	487 (4s)
(1000, 200)	2	532 (2700s)	535 (260s)	532 ($3600s^{to}$)	535 (450s)

Table 2: Different variants of our algorithm. ~~Crossing~~ letters *s* and *h* indicates whether our search strategy and/or the heuristic cut is used.

$(\lvert A\rvert,\lvert S\rvert)$	**k=1**				**k=2**				**k=3**			
	#o	g	i	t	#o	g	i	t	#o	g	i	t
(30, 11)	7	6.5%	15, 5	1, 0	8	2.7%	60, 20	3.5, 1	10	0	190, 94	20, 9
(150, 30)	5	3.6%	15, 5	11, 7	3	3.0%	159, 50	1312, 49	3	2.%	1324, 455	5515, 918

Table 3: Analysis of the cuts on small and medium benchmarks (ten instances of each). Legend: *#o* - number of optimal solutions by the heuristic cut; *g* - average optimality gap for suboptimal values; *i* and *t* - average number of iterations and execution time by the complete and heuristic variant.

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 $\langle p, q\rangle$-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 $\langle p, q\rangle$-recoverability in the case where $q = 0$, i.e., when no accumulative cost is allowed at all; this in 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 $\Sigma_3^{\mathbf{P}}$-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 β. Probabilistic weighted variants, named *(α, β)-weighted* super solutions, were introduced [13], where a is substituted by α, the breakage probability, and β, 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 $\Sigma_3^{\mathbf{P}}$-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

[1] J. E. Beasley. 1990. OR-Library: distributing test problems by electronic mail. *Journal of the operational research society* (1990), 1069–1072.
[2] D. Bergman and A.A. Ciré. 2016. Multiobjective Optimization by Decision Diagrams. In *Proceedings of CP'16*. 86–95.
[3] Miquel Bofill, Dídac Busquets, Víctor Muñoz, and Mateu Villaret. 2013. Reformulation based MaxSAT robustness. *Constraints* 18, 2 (2013), 202–235.
[4] M. Bruneau. 2003. A Framework to Quantitatively Assess and Enhance the Seismic Resilience of Communities. In *Earthquake Spectra*, Vol. 19.
[5] C. Crawford, Z. Rahaman, and Sandip Sen. 2016. Evaluating the Efficiency of Robust Team Formation Algorithms. In *AAMAS'16 Workshops - Best Papers - Revised Selected Papers*. 14–29.
[6] T. Feydy and P. J. Stuckey. 2009. Lazy Clause Generation Reengineered. In *Proceedings of CP'09*. 352–366.
[7] C. Gao, X. Yao, T. Weise, and J. Li. 2015. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research* 246, 3 (2015), 750–761.
[8] M. R. Garey and D.S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.
[9] J. M. George, J. Pinto, P. B. Sujit, and J. B. Sousa. 2010. Multiple UAV coalition formation strategies. In *Proceedings of AAMAS'10*. 1503–1504.
[10] Matthew L. Ginsberg, Andrew J. Parkes, and Amitabha Roy. 1998. Supermodels and Robustness. In *Proceedings of AAAI'98, IAAI'98*. 334–339.
[11] E. Hebrard, B. Hnich, and T. Walsh. 2004. Robust Solutions for Constraint Satisfaction and Optimization. In *Proceedings of ECAI'04*. 186–190.
[12] E. Hebrard, B. Hnich, and T. Walsh. 2004. Super Solutions in Constraint Programming. In *Proceedings of CPAIOR'04*. 157–172.
[13] A. Holland and B. O'Sullivan. 2005. Weighted Super Solutions for Constraint Programs. In *Proceedings of AAAI'05, IAAI'05*. 378–383.
[14] C. Holling. 1973. Resilience and Stability of Ecological Systems. *Annual Review of Ecology and Systematics* 4 (1973), 1–23.
[15] S. Kadioglu and M. Sellmann. 2009. Dialectic Search. In *Proceedings of CP'09*. 486–500.
[16] M. Kargar, A. An, and M. Zihayat. 2012. Efficient Bi-objective Team Formation in Social Networks. In *Proceedings of ECML-PKDD'12*. 483–498.
[17] H. Kitano and S. Tadokoro. 2001. RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. *AI Magazine* 22, 1 (2001), 39–52.
[18] T. Lappas, K. Liu, and E. Terzi. 2009. Finding a team of experts in social networks. In *Proceedings of SIGKDD'09*. 467–476.
[19] T. Matthews, S. D. Ramchurn, and G. Chalkiadakis. 2012. Competing with Humans at Fantasy Football: Team Formation in Large Partially-Observable Domains. In *Proceedings of AAAI'12*. 1394–1400.
[20] N. Musliu. 2006. Local Search Algorithm for Unicost Set Covering Problem. In *Proceedings of IEA/AIE'06*. 302–311.
[21] O. Ohrimenko, P. J. Stuckey, and M. Codish. 2009. Propagation via lazy clause generation. *Constraints* 14, 3 (2009), 357–391.
[22] T. Okimoto, T. Ribeiro, D. Bouchabou, and K. Inoue. 2016. Mission Oriented Robust Multi-Team Formation and Its Application to Robot Rescue Simulation. In *Proceedings of IJCAI'16*. 454–460.
[23] T. Okimoto, N. Schwind, M. Clement, T. Ribeiro, K. Inoue, and P. Marquis. 2015. How to Form a Task-Oriented Robust Team. In *Proceedings of AAMAS'15*. 395–403.
[24] C. M. Papadimitriou. 1994. *Computational complexity*. Addison-Wesley, Reading, Massachusetts.
[25] T. Rahwan, T. P. Michalak, M. Wooldridge, and N. R. Jennings. 2015. Coalition structure generation: A survey. *Journal of Artificial Intelligence* 229 (2015), 139–174.
[26] N. Schwind, M. Magnin, K. Inoue, T. Okimoto, T. Sato, K. Minami, and H. Maruyama. 2016. Formalization of resilience for constraint-based dynamic systems. *Journal of Reliable Intelligent Environments* 2, 1 (2016), 17–35.
[27] J. P. M. Silva, I. Lynce, and S. Malik. 2009. Conflict-Driven Clause Learning SAT Solvers. In *Handbook of Satisfiability*, A. Biere, M. Heule, H. Maaren, and T. Walsh (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press, 131–153.
[28] J. P. M. Silva and K. A. Sakallah. 1999. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Trans. Computers* 48, 5 (1999), 506–521.
[29] T. Stidsen, K. A. Andersen, and B. Dammann. 2014. A Branch and Bound Algorithm for a Class of Biobjective Mixed Integer Programs. *Management Science* 60, 4 (2014), 1009–1032.