2F2-2

Building Teams Resilient to Change

Nicolas Schwind[*1] Emir Demirović[*2] Tenda Okimoto[*3] Katsumi Inoue[*4*5*6]

[*1] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

[*2] TU Wien, Vienna, Austria [*3] Kobe University, Kobe, Japan

[*4] National Institute of Informatics, Tokyo, Japan

[*5] SOKENDAI (The Graduate University for Advanced Studies), Tokyo, Japan

[*6] School of Computing, Tokyo Institute of Technology, Tokyo, Japan

Team formation (TF) consists in finding the least expensive team of agents such that a certain set of skills is covered. We formally introduce here *recoverable team formation* (RTF), a generalization of TF, which accounts for the dynamic nature of the environment: some selected agents may unexpectedly become unavailable due to failure or illness, and one is allowed to select afterwards additional agents to (re)-cover the set of skills. We analyze the computational complexity of RTF, provide both complete and heuristic algorithms, and evaluate their performance.

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, Unmanned Aerial Vehicle operations, social networks, online football prediction games, among others. Moreover, TF is closely related to set covering and hitting sets, 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, that is, how *resilient* the team is.

One aspect of resilience for TF has been introduced in [8], 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 a team to keep performing without taking any measures after unfortunate changes happen. Interestingly, the computational complexity of robust team formation is not higher than for TF: both are **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 [3, 6, 10], 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 [10], see our related work section) and recoverability. In TF, recoverability consists in considering an additional cost that might need to be paid to restore the team's efficiency if k agents are removed from the team. Our notion of recoverability here can be viewed as a generalization of the notion of robustness, i.e., both notions coincide when restoring the team's initial efficiency must not incur a cost. Considering recoverability allows for more flexibility: 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.

The flexibility of our framework is paid by the increase of the computational complexity - from **NP**- to $\mathbf{\Sigma_3^P}$-hardness. Thus, developing algorithms for recoverable team formation (RTF) is harder than for TF and conventional techniques for TF cannot 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. We also propose a heuristic cut to provide a trade-off between computational time and solution quality.

2. (Robust) Team Formation

We give preliminaries on (robust) team formation. Basic definitions and theorems are provided and we direct the interested reader to [8] for more details and proofs.

Definition 1. *(TF Problem Description) A TF problem description is a tuple* $TF = \langle A, S, f, \alpha \rangle$ *where* $A = \{a_1, a_2,$

*1 In the literature sometimes the term *task* [8] is used instead of *skill*, but the problem remains the same.

	a_0	a_1	a_2	a_3	a_4	a_5
$\alpha(a_i)$	$\{s_0, s_1\}$	$\{s_1, s_2\}$	$\{s_0, s_2\}$	$\{s_2\}$	$\{s_0\}$	$\{s_1\}$
$f(a_i)$	5	5	4	3	3	3
$h(a_i)$	10	10	8	3	3	$+\infty$

Table 1: Illustrative example definition.

..., 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 so-called cost function*, and α is a mapping from A to 2^S. $T \subseteq A$ is called a* team*.*

For simplicity we assume $f(T) = \sum_{a_i \in T} f(a_i)$*[2] when $T \neq \emptyset$, and $f(\emptyset) = 0$.

Definition 2. *Team T is said to be c-costly if $f(T) \leq c$.*

Definition 3. *Team T is efficient if $S = \bigcup_{a_i \in T} \alpha(a_i)$.*

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 1. *[8]* SETF is **NP**-complete.

Definition 4. *[8] 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. *[8]* 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: in addition to the team's initial cost we consider an additional cost called *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 5. *(Recoverable Team Formation Problem Description) A RTF problem description is a tuple $RTF = \langle A, S, f, \alpha, h\rangle$ 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, $f : 2^A \to \mathbb{N} \cup \{+\infty\}$ is the so called* first-step *cost function, α is a mapping from A to 2^S, and $h : 2^A \to \mathbb{N} \cup \{+\infty\}$ is the so-called* second-step *cost function.*

We now introduce the main new property.

*2 Abusing notations, when a single agent a_i is considered, we often use the notation $f(a_i)$ instead of $f(\{a_i\})$

Team	$T_1 = \{a_0, a_1, a_2\}$	$T_2 = \{a_0, a_1\}$	$T_3 = \{a_3, a_4, a_5\}$
$f(T)$	14	10	9
rec. cost	0	3	10
total cost	14	13	19

Table 2: Example for three teams and $k = 1$.

Definition 6. *(Team Recoverablility) Given a RTF problem description $RTF = \langle A, S, f, \alpha, h\rangle$ and an efficient team T, we say that T is* $\langle k, c_2\rangle$-recoverable *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 case $f(T) \leq c_1$, then T is said to be $\langle c_1, k, c_2\rangle$-recoverable.*

Similarly as before, we have assume that $f(T) = \sum_{a_i \in T} f(a_i)$ and $h(T) = \sum_{a_i \in T} h(a_i)$ when $T \neq \emptyset$, and $f(\emptyset) = h(\emptyset) = 0$. The recovery cost of a team is computed as follows: for every removal of k agents from the team, we calculate the minimum second-step cost (given by h) necessary to restore the lost efficiency and then take the maximum out of all values. Intuitively, this represents the amount that needs to be paid in the worst case when k agents are removed from the team. While robustness requires a team that still remains efficient after any k agents are removed, recoverability searches for a team that is "harmless" to repair if k agents are lost.

Let us consider the RTF problem with $k = 1$, i.e., where the recovery cost is analyzed when removing a single agent from a team. The set of agents and skills are defined as $A = \{a_0, \ldots, a_5\}$ and $S = \{s_0, s_1, s_2\}$. Table 1 depicts the function α that maps every agent $a_i \in A$ to a subset of S and its costs associated through the first- and second-step functions f and h. The total cost is the sum of both costs.

The first-step cost of the team $T_1 = \{a_0, a_1, a_2\}$ is computed as: $f(T_1) = f(a_0) + f(a_1) + f(a_2) = 14$. T_1 is 1-robust, i.e., it remains efficient regardless of which single agent is removed from T_1. To determine the recovery cost of the second team T_2, we take into account every possible agent removal from T_2. Removing the agent a_0 from T_2 would result in a team which does not have skill s_0 covered. To restore its efficiency, the best way to do so would be to add the agent a_4, resulting in a recovery cost of $h(a_4) = 3$. Likewise, removing a_1 would lead to a recovery cost of $h(a_3) = 3$. The recovery cost of the team is then calculated as: $\max(h(a_3), h(a_4)) = 3$. The first team is robust, but initially expensive ($f(T_1) = 14$). The total cost of T_2 is lower than for T_1, making it appealing if robustness is not necessary. The third team has low initial cost and high recovery cost. There is no agreed notion of "best team" here since it depends on the concrete application, users, budgets, and risk management strategies. Providing multiple teams to choose from, each with its own unique properties, allows for better decision making.

We show that RTF is a generalization of robust TF.

Proposition 1. *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.*

4. Computational Complexity

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

Definition 7. *(RTF problem)*

- **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 [9] for more details). Higher complexity classes are defined using oracles. In particular, $\mathbf{\Sigma_2^P} = \mathbf{NP^{NP}}$ 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 $\mathbf{\Sigma_3^P}$ is the class of decision problems that are solved in non-deterministic polynomial time by deterministic Turing machines using an oracle for $\mathbf{\Sigma_2^P}$ in polynomial time. We get the following complexity result:

Proposition 2. RTF is $\mathbf{\Sigma_3^P}$-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 recovery cost. The number of combinations is exponential with respect to k, and for each removal computing the 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 3. *For* $k \geq 0$, k-RTF *is* **NP***-complete.*

5. Algorithms

We now describe our complete and heuristic algorithms. The objective is to minimize the sum of the first- and second-step costs. An outline is given in Algorithm 1. Iteratively, a single team is considered (line 4) ($T_{cur} \subseteq A$) and its recovery cost is determined (lines 5-10). The recovery cost is calculated by enumerating subsets a_k of k agents from T_{cur}. For each subset a_k, a recovery cost specific to that set is calculated by removing a_k from T_{cur} and then computing the recovery team $T_{rec} \subseteq A \setminus T_{cur}$ with minimum cost with respect to the second-step cost function h such that $T_{cur} \setminus a_k \cup T_{rec}$ is efficient (line 7). The highest value among computed recovery costs for the subsets a_k is taken to be the recovery cost for team T_{cur}. The team T_{cur} is then recorded as the best team found if its costs are lower than the previous best (line 11-14). Lastly, a key step is executed: cuts are generated to prune teams that share a similar substructure as T_{cur}. The choice of cuts determines whether the algorithm is (in)complete.

We introduce the following notation: $C_2(T)$ is second-step cost of T, $C_{total}(T) = f(T) + C_2(T)$, $C_2(T, a_k)$ is the

Algorithm 1: Algorithm Outline for RTF

```
   input: RTF = {A, S, f, α, h}
   ouput: T_best with min cost (f(T_best) + C_2(T_best))
1  begin
2      T_best ⟵ ∅; c_1 ⟵ +∞; c_2 ⟵ +∞
3      while not all feasible teams have been explored
        do
4          T_cur ⟵ selectNewTeam() // TF
5          c_rec^max ⟵ 0; a_k^* ⟵ ∅
6          foreach a_k ∈ {a : a ⊆ T_cur ∧ |a| = k} do
               // calculate the minimum recovery cost wrt
                  removals of a_k
7              c_rec = minRecoveryCost(T_cur, a_k, RTF)
8              if c_rec ≥ c_rec^max then
9                  c_rec^max ⟵ c_rec
10                 a_k^* ⟵ a_k
11         if f(T_cur) + c_rec^max < c_1 + c_2 then
12             c_1 ⟵ f(T_cur)
13             c_2 ⟵ c_rec^max
14             T_best ⟵ T_cur
           // prune search space based on team T
15         generateCut(T_cur, a_k^*, T_best, RTF)
16     return T_best
```

second-step cost associated for the specific case when agents in a_k are removed from T, $M(S)$ is the set of agents that posses at least one of the skills in S, and $S(T, a_k)$ is the set of skills that become *uncovered* when agents in a_k are removed from T, T_{best} is the best team found so far during the search, and lastly $C_2^{max}(T, T_{best}) = C_{total}(T_{best}) - f(T) - 1$ is the threshold value of the recovery cost for team T in order to be considered better than T_{best}. Recall that computing $C_2(T, a_k)$ amounts to solving the team formation problem with the skills $S(T, a_k)$ and agents $M(S(T, a_k))\backslash T$. Let T_{cur} be the currently analyzed team, a_k^* be the set of agents such that $C_2(T_{cur}, a_k^*) > C_2^{max}(T_{cur}, T_{best})$, and T is any future team we might consider. We now present the cut:

$$C_2^{max}(T, T_{best}) < C_2(T_{cur}, a_k^*) \Rightarrow \sum_{i \in M(S(T, a_k^*))\backslash a_k^*} x_i \geq 1 \quad (1)$$

The cut forces the inclusion of at least one agent from $M(S(T, a_k^*))\backslash a_k^*$ under the condition that the left-hand side is satisfied, since otherwise the resulting recovery cost will be greater than the threshold value. Essentially, the partial assignment of T_{cur} concerning agents from $M(S(T, a_k^*))\backslash a_k^*$ is responsible for the high recovery cost. No solution better than T_{best} is pruned, hence the algorithm is complete.

The resulting cut is nonlinear. Traversing the search space by considering teams in increasing first-step cost allows us to omit the left-hand side, as it becomes redundant (always satisfied), effectively linearizing the cut. Linear cuts are typically easier to solve in practice.

Our heuristic cut provides means of reducing the computational time at the expense of possibly losing optimality:

$$-\sum_{i \in a_k^*} x_i \geq 1 - |a_k^*| \quad (2)$$

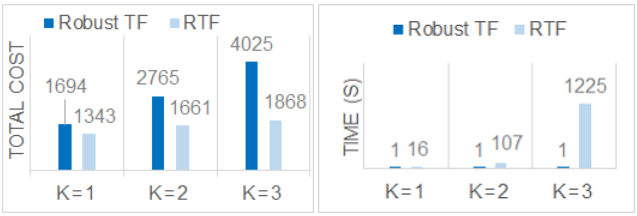

Figure 1: Comparison on instances with 150 agents and 30 skills. Average for ten instances presented.

The intuition is that the removal agents a_k^* are the cause of high recovery costs, so by removing all teams with a_k^* the same problem is not encountered again. In the experimental section, we show that good solutions can still be obtained when using this cut. The heuristic cut is generated in addition to the base cut in our heuristic algorithm.

6. Experimental Results

The goal of our experimentation is to show that a recoverable team is cheaper than a robust one, by is harder to compute. Yet solutions for reasonably sized teams can be found, despite RTF being a $\mathbf{\Sigma_3^P}$-hard problem. The results are presented in Figure 1 and Table 3.

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 [8]. The medium ones (from 100 to 150 agents and 20 to 30 skills) are set covering instances used in [2, 11]. 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., [5, 7]). In our experiments we set $h = f$ (recall Definition 6). All tests were performed on an Intel Core i3-2330M CPU @ 2.20GHz with 4 GB RAM using a computation limit of one hour. Our algorithm is implemented in C++. To compute solutions for robust TF, we used *Cplex* and integer programming, as it proved to be significantly faster than the previous approaches proposed in [4] and [8].

The comparison with robustness is shown in Figure 1 for instances with 150 agents and 30 skills (the values are averaged over ten instances). The small instances from [8] are solved within seconds for $k = \{1, 2, 3\}$. Overall, RTF has a significantly lower cost than robust TF for the same value of k, and the gap increases as k grows. 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. This is to be expected given Proposition 3. The heuristic cut (Equation 2) provides means to combat the increase in computational time (see next section).

We considered adding the heuristic cut (Equation 2) in addition to the base cut (Equation 1), and our special search strategy (going through teams in increasing first-step cost). We present detailed results in Table 3 on a few instances and similar results are obtained for other instances. Overall, both our search strategy and our heuristic cut become more important as the size of the instance and k grow, reducing

($\|A\|$,$\|S\|$)	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 3: Different variants of our algorithm. ~~Crossing~~ letters s and h indicates whether our search strategy and/or the heuristic cut is used.

the execution time while maintaining high solution quality. We attribute this behavior to the linearization of the cuts and the search space reduction, as discussed in Section 5.

7. Conclusion

We introduced recoverable team formation (RTF), a novel problem where in addition to the team's cost we examine the cost related to restoring functionality after k agents have been removed. We provided a framework for building teams resilient to change, which is more general than the work in [8], at the expense of a complexity shift to $\mathbf{\Sigma_3^P}$-hardness. Using our proposed cuts and search strategy, our algorithms were able to solve reasonably sized problems despite the high computational complexity. Our results have shown the drastic difference in cost between robustness and recoverability, in favor of recoverability. Computing multiple teams, stochastic settings, and introducing recoverability to other problems are all topics for future work.

References

[1] J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the operational research society*, pages 1069–1072, 1990.

[2] D. Bergman and A.A. Ciré. Multiobjective optimization by decision diagrams. In *Proc. of CP-16*, pages 86–95, 2016.

[3] M. Bruneau. A framework to quantitatively assess and enhance the seismic resilience of communities. In *Earthquake Spectra*, volume 19, 2003.

[4] C. Crawford, Z. Rahaman, and Sandip Sen. Evaluating the efficiency of robust team formation algorithms. In *AAMAS-16 Workshops - Best Papers - Revised Selected Papers*, pages 14–29, 2016.

[5] C. Gao, X. Yao, T. Weise, and J. Li. An efficient local search heuristic with row weighting for the unicost set covering problem. *European Journal of Operational Research*, 246(3):750–761, 2015.

[6] C. Holling. Resilience and stability of ecological systems. *Annual Review of Ecology and Systematics*, 4:1–23, 1973.

[7] S. Kadioglu and M. Sellmann. Dialectic search. In *Proc. of CP-09*, pages 486–500, 2009.

[8] T. Okimoto, N. Schwind, M. Clement, T. Ribeiro, K. Inoue, and P. Marquis. How to form a task-oriented robust team. In *Proc. of AAMAS-15*, pages 395–403, 2015.

[9] C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

[10] N. Schwind, M. Magnin, K. Inoue, T. Okimoto, T. Sato, K. Minami, and H. Maruyama. Formalization of resilience for constraint-based dynamic systems. *Journal of Reliable Intelligent Environments*, 2(1):17–35, 2016.

[11] T. Stidsen, K. A. Andersen, and B. Dammann. A branch and bound algorithm for a class of biobjective mixed integer programs. *Management Science*, 60(4):1009–1032, 2014.