Evaluating the Efficiency of Robust Team Formation Algorithms

Chad Crawford, Zenefa Rahaman, and Sandip Sen

University of Tulsa
{chad-crawford,zenefa-rahaman,sandip-sen}@utulsa.edu

Abstract. Selecting a minimal-cost team from a set of agents, with associated costs, to complete a given set of tasks is a common and important multiagent systems problem. Often, some degree of fault-tolerance in such teams is also required which enables the team to continue to complete all tasks even if a subset of the agents are incapacitated. A k-robust team is one that is capable of completing all assigned tasks when any k team members are not available. The corresponding decision problem of selecting a k-robust team that costs no more than a desired cost threshold has been shown to be NP-Complete. We present and experimentally evaluate, for varying problem sizes and characteristics, heuristic and evolutionary approximation approaches to find optimal-cost k-robust teams which can be used for large problems. We present a Linear Programming approximation algorithm that is found to produce optimal results for small problem sizes, and prove that it is a $2\ln(n+k) + O(\ln n)$-factor approximation. We also present three heuristic algorithms and an evolutionary computation approach which scales up to larger problems. Another advantage of the evolutionary scheme is that can approximate the Pareto-frontier of teams trading off robustness and cost.

Keywords: Robust teams, Approximation algorithms, Set multicover algorithms

1 Introduction

Team formation is a critical activity in multiagent systems, particularly in cooperative settings. In an often-studied scenario, teams are formed in a distributed setting where demands of the situation necessitates otherwise self-interested agents to join together in cooperative teams or coalitions [19]. A significant amount of work in the cooperative game theory and in the multiagent systems community on coalition formation addresses this scenario. These approaches seek fair payoff distribution schemes between team members to incentivize agents joining and staying in stable coalitions.

Teams can be also formed in a centralized manner with prior knowledge of the capabilities and expertise requirement in the domain. A centralized decision maker is provided requirements for fulfilling a set of tasks in a domain, where each task has certain resource and/or capability requirements, using a set of

available agents with given capabilities and resource capacities. Typically the emphasis in such a scenario (e.g., forming a team of robots to explore inhospitable environments) is to guarantee a degree of certainty of outcomes while minimizing cost. Many real life scenarios can be framed as an instance of this team formation problem, e.g., choosing team of first responders, surveillance and reconnaissance missions with unmanned vehicles, robotics assembly tasks, assisting disabled patients with chores at their homes, etc.

The problem of selecting a minimal cost team of agents with the necessary capabilities to achieve a given set of tasks can be posed as an optimization problem. Recently, a variant of this problem has been studied which examines the robustness of such teams. Robustness is defined by the number of team members, k, that may be lost without compromising the ability of the rest of the team members achieving all tasks [17]. The loss of team members is prevalent particularly with robot teams, where robots may become physically incapacitated or isolated, lose power or connectivity, get disoriented, or be impaired by software errors. Such incapacitated members can no longer contribute to completing team tasks and can therefore limit the effectiveness of the team. The corresponding Task-Oriented Robust Team Formation (TORTF) optimization problem can have variants: (A) given a cost budget, finding the team which is robust against the loss of the largest number of agents, i.e., a k-robust team with the maximal k value, (B) given a desired robustness level k, finding a k-robust team of minimal cost, and (C) a dual-objective constraint optimization problem with a pareto-frontier of solutions trading off team cost and team robustness. Though the corresponding decision problems are NP-complete, Okimoto *et al.* present an exact algorithm for solving variant C and evaluates it on small problem sizes [17]. The TORTF problem can be mapped into the Weighted Set Multicover (SMC) problem, which has been widely studied in the theoretical computer science literature. A variety of exact algorithms have been proposed for the SMC [8], all of which understandably are of exponential complexity and hence cannot be used for large problem sizes.

In this paper we present our work on developing and evaluating approximation algorithms for variation B of the TORTF optimization problem, i.e., the problem of generating the minimal-cost k-robust team. We present a suite of approximation algorithms for this problem: three greedy heuristic algorithms, a genetic algorithm based optimization approach and a Linear Programming based approximation for the SMC problem and prove that it is a $2\ln(n+k)+O(\ln n)$-factor approximation. The genetic algorithm variant, NSGA-II [5], is designed to solve Multi-Objective Optimization problems and hence can be used to address variant C of the TORTF problem. We evaluate the relative efficiencies of these optimization techniques on a testbed of TORTF problems varying the number of agents, the number of tasks, the tasks/agent ratio, the cost distributions of the agents, etc. We present an in-depth analysis of the experimental results, including scalability considerations, and present guidelines for choosing an algorithm given problem characteristics.

2 Problem Definition

Each agent in a TORTF scenario is capable of doing a subset of the needed tasks. Teams are groups of agents that can complete a task that any member can complete. Each agent has an associated cost for adding that agent to the team. The goal of TORTF is to find a team of minimal cost such that if any k agents are removed from the team, it retains the capability of completing all assigned tasks. In practical team formation scenarios, there are often cases where some members may be incapacitated and may not be able to complete some or many of the tasks required of them. In those scenarios, other agents are expected to step up and complete the tasks in their place.

A TORTF problem can be specified as the tuple ($\mathcal{A}$, $\mathcal{T}$, c, α, k), where $\mathcal{A} = \{a_1, \ldots, a_n\}$ is a set of agents, $\mathcal{T} = \{t_1, \ldots t_m\}$ is a set of tasks, $c : \mathcal{A} \to \mathbb{R}^+$ is the cost function assigning a cost to each agent, and $\alpha : \mathcal{A} \to 2^{\mathcal{T}}$ is the function that lists the subset of tasks that an agent can complete. Unlike other task assignment problems, there are no time/resource constraints, so an agent is expected to complete all tasks that they are capable of doing.

Furthermore, if $S \subseteq \mathcal{A}$ is a team of agents, then the total cost of forming the team is $\mathrm{c}(S) = \sum_{a \in S} c(a)$.

Definition 1 (Efficiency). *[17] defines a team $S \subseteq \mathcal{A}$ to be* **efficient** *if and only if the team members as a group can complete all tasks:*

$$\bigcup_{a \in S} \alpha(a) = \mathcal{T}.$$

We will prefer to use the term robust *or* effective *to describe this property, since we are also interested in the efficiency of the algorithms in this paper.*

Definition 2 (k-robustness). *A team, $S \subseteq \mathcal{A}$, is* **k-robust**, *where $k \in \{0, \ldots, n-1\}$, if and only if it is efficient when any k agents are removed from the team:*

$$\forall S' \subseteq S, |S'| \leq k \Rightarrow S - S' \text{ is efficient.}$$

Furthermore,

$$\text{k-robust}(S) = \max\{k | S \text{ is } k\text{-robust}\}$$

is robustness of a team – the maximum number of agents which can be removed from the team without violating efficiency. We define the robustness of a team for a single task to be the number of team members that can perform the task:

$$\text{k-robust}_j(S) = |\{a_i \in S | t_j \in \alpha(a_i)\}|$$

Computing k-robust(S) can be done in $O(n)$ time, as explained in [17]. The procedure computes the number of agents that can complete each task in $\mathcal{T}$; the k-robustness is then the minimum number of agents assigned to some task minus 1. We examine two optimization problems: (a) Find the least-cost k-robust team, and (b) compute the Pareto frontier of teams that minimize cost and maximize k.

3 Related Work

Team formation is a critical problem for multiagent domains that require agent cooperation. Research on team formation has been focused on applications such as robotics [12], engineering projects [3] or business projects [21]. Deciding an optimal team for a project can also have different types of constraints, for example, social ties or compatibility between pairs of agents [13]. Another approach by [15] examines teams that collaborate to vote to make decisions. They compare teams of informed, but not diverse voters against more uninformed but diverse voters: their results indicate that diverse teams are more effective. One factor that needs to be accounted for in team formation is the possibility that some agents will not be able to contribute to the group after the team is selected due to unforeseen circumstances.

Several papers focus on coalitional skill games, for example, Bachrach *et al.* [2] gives a simple model of cooperation among agents. Each agent has skills and each task requires a set of skills; it is only possible to complete the tasks if the coalition's agents cover the set of required skills for the task. [1] focuses on Coalition Structure Generation (CSG), which partitions a set of agents into groups to maximize the sum of the values of all coalitions. Finding an optimal coalition structure is NP-hard.

The k-robustness problem is an agent-based formulation of the weighted set multicover problem. The set cover problem was one of the original 21 problems to be proven NP-complete by Karp [11], and the natural greedy approximation was proven to be within a factor of H_n (the partial sum of the first n terms of the Harmonic series) for the unweighted problem in [10, 14] and for the weighted problem in [4]. Furthermore, it has been proven that there cannot exist an efficient approximation better than $\ln n$ for the set cover [6] (the aforementioned greedy mechanism has this factor since $H_n = \ln n + O(1)$). The weighted set multicover problem is proven to be NP-hard [17, 20].

A number of exact algorithms have been developed to solve the weighted set multi-cover problem [7, 9, 8, 17]. The algorithm developed in [17] calculates the entire Pareto frontier of solutions using a branch-and-bound approach. However, this algorithm is unusable for even moderately large number of agents or tasks. The fastest exact algorithm discussed in [8] uses a dynamic programming approach to build multiset covers that builds up a solution from combinations of multisets. This algorithm runs in $O((k+2)^n)$ time, which is the fastest known optimal algorithm that we know of.

The natural greedy algorithm for weighted set cover yields an H_n-factor approximation. Similar approximations have been found for the unweighted/weighted set multicover problems [20]. The approximation factor is derived by showing that the "prices" assigned to tasks by the greedy algorithm satisfy the dual of the integer programming formulation of the problem, where the dual represents feasible lower bounds on the true cost of the optimal allocation. Since the greedy approximation satisfies the dual and since the dual of the LP relaxation is within a factor of H_n of the integer solution, the greedy approximation is also within a factor of H_n.

4 Approximations of the TORTF problem

We now present the approximation approaches for the TORTF problem that we evaluate in this paper. We assume that for each of these algorithms, for the given k, a k-robust team exists. It is simple to verify existence: since a k robust team will also be L-robust for any $L \leq k$, then a k-robust team exists if and only if k-robust$(\mathcal{A}) \geq k$.

4.1 Greedy algorithms

We first describe a set of greedy approaches to approximate solutions to a TORTF problem.

Greedy Heuristic 1: We present (see Algorithm 1) the naive greedy approach as one method for selecting a robust team G. This algorithm has been studied extensively for the weighted set multi-cover problem, and has been proven to be an H_n-factor approximation of the optimal solution [20]. This algorithm and its derivative, Heuristic 2, run in $O(n^2)$ time.

This selects the agent with minimum *price*, which is the cost of the agent per needed task it can complete. In other words, the greedy algorithm iteratively selects the cheapest agent given the number of additional tasks that they can do and will help the team achieve the specified level of robustness. This is repeated until the team is k-robust.

Algorithm 1 Greedy Heuristic 1

procedure AGENT-TEAM($\mathcal{A}, \mathcal{T}, c, \alpha, k$)
  $G \leftarrow \emptyset$
  while k-robust$(G) < k$ **do**
    $C \leftarrow \{t_j \in \mathcal{T} |\, \text{k-robust}_j(G) < k\}$
    Choose $a^* \leftarrow \arg\min_{a \in \mathcal{A}} \frac{c(a)}{|\alpha(a) \cap C|}$
    $G \leftarrow G \cup \{a^*\}$
  return G

Greedy Heuristic 2: This modification of the first greedy approach focuses on tasks which can be completed by the fewest number of agents. Unlike the last method, which considered the entire set $\mathcal{A}$ when choosing the best agent, this mechanism only considers agents that can perform the task t^*, which corresponds to the task with the fewest number of unselected agents that can complete it. Our motivation behind this approach is to minimize losses on the most restrictive choices. As the first greedy heuristic progresses, it will need to find k agents to complete t^*. However, as more and more agents are selected, the number of

unselected agents capable of completing t^* becomes smaller. To alleviate this issue, this variation of the greedy heuristic selects t^* to cover immediately and continues to make similarly greedy choices subsequently (see Algorithm 2 for the complete algorithm).

Algorithm 2 Greedy Heuristic 2

procedure AGENT-TEAM($\mathcal{A}, \mathcal{T}, c, \alpha, k$)
  $G \leftarrow \emptyset$
  while k-robust(G) $< k$ **do**
    $C \leftarrow \{t_j \in \mathcal{T} | \text{ k-robust}_j(G) < k\}$
    $t^* \leftarrow \arg\min_{t \in C} |a \in \mathcal{A} \setminus G : t \in \alpha(a)|$
    $TA \leftarrow \{a \in \mathcal{A} \setminus G : t^* \in \alpha(a)\}$
    Choose $a^* \leftarrow \arg\min_{a \in TA} \frac{c(a)}{|\alpha(a) \cap C|}$
    $G \leftarrow G \cup \{a^*\}$
  return G

Greedy Heuristic 3: In this heuristic, agents are selected based on their marginal utility. The marginal utility is the difference between the benefit of a team with that agent and without the agent. The algorithm (see Algorithm 3) greedily selects agents at each turn, always choosing the agent that maximizes marginal utility given the previously selected set of agents. Let $\tau(G) = \{t_j \in \mathcal{T} : \text{k-robust}_j(G) < k\}$ is the set of *uncovered* tasks not k-covered by G. The marginal utility for an agent a, given the currently selected set of team members G, is approximated as

$$u(a, G) = \sum_{t \in \alpha(a)} \binom{z_t(\mathcal{A} \setminus G) - 1}{r_t(\mathcal{A} \setminus G)} \cdot \bar{c}_t(G \cup \{a\}) - \left(\text{c}(a) + \sum_{t \in \alpha(a)} \binom{z_t(\mathcal{A} \setminus G) - 1}{r_t(\mathcal{A} \setminus G) - 1} \cdot \bar{c}_t(G \cup \{a\}), \right) \quad (1)$$

where z_t is the number of agents available to complete an uncovered task t and r_t is the additional robustness needed for that task ($k - \text{k-robust}_t(G)$) given the set of agents G chosen so far. The function $\bar{c}$ is the sum of the average costs per uncovered task of agents that can complete t:

$$\bar{c}_t(G) = \sum_{a : a \in \mathcal{A} \setminus G \text{ and } t \in \alpha(a)} \frac{c(a)}{|\alpha(a) \cap \tau(G)|}. \quad (2)$$

The first term in the expression for u is the average potential utility when a is not added to the team, i.e., it is the "average" cost of r_t randomly selected agents completing the task. The second term computes how much potential cost is added if we first select a and then select the rest of the team.

Algorithm 3 Greedy Heuristic 3

procedure AGENT-TEAM($\mathcal{A}, \mathcal{T}, c, \alpha, k$)
  $G \leftarrow \emptyset$
  while k-robust(G) $< k$ **do**
    Choose $a^* \leftarrow \arg\max_{a_i \in \mathcal{A} \setminus G} u(a_i, G)$
    $G \leftarrow G \cup \{a^*\}$
  return G

4.2 Genetic Algorithm

The team formation problem can be framed as a multiple-objective optimization problem: maximizing k and minimizing cost. While previous algorithms compute the solution for only one k, knowing solutions for other k might help find better teams at k-robustness. We apply a multi-objective genetic algorithm to compute the entire Pareto frontier of solutions to the TORTF problem. That is, this GA will not be given a single k value to optimize, but will instead attempt to learn team allocations to ensure k-robustness for any k.

We adapt the implementation of the Nondominated Sorting Genetic Algorithm (NSGA-II) [5]. The NSGA-II algorithm tracks both a parent and child population. To optimize solutions on the Pareto frontier, NSGA-II adds a preprocessing step to each generation: the population is broken into non-dominating fronts, sorted by the domination relation. Another metric further sorts the solutions inside each front based on their diversity; this encourages the GA to select solutions across the entire Pareto frontier. The first N solutions among this group are put inside the *parent population*. The following M solutions compose the *child population*, and are generated using GA operators on the parent population. We are interested in solving the general optimization problem, but solutions for one k value may help inform solutions for other k values, assuming that the best teams with varying robustness would share some overlap in members. The NSGA-II algorithm is ideal since it is designed to find diverse solutions across the Pareto frontier.

NSGA-II is much slower than the heuristic and linear programming approaches. The time complexity of the preprocessing step is $O(2N^3)$, where N is the size of the chromosome pool. While this is faster than the naive approach to sorting the pool by multiple objectives, it runs significantly longer than the other algorithms presented.

Chromosomes in our GA are represented as bitstrings; if $b = b_1, \ldots, b_n$ is a bitstring, then agent $a_i \in \mathcal{A}$ belongs to the corresponding team iff $b_i = 1$. Our algorithm uses Tournament selection with $p = 0.65$; when generating the pool of chromosomes for crossover, it randomly selects two chromosomes from the parent pool and places the better chromosome in the new pool with probability p, otherwise it adds the worse chromosome. Crossover is uniform, so each bit has an equally likely probability of being copied from the first or second parent. The probability of crossover occuring is 95%, otherwise a parent is directly copied.

Bitwise mutation is then performed, in which a randomly chosen bit is "flipped" with a mutation probability of 10% per chromosome in the newly generated child pool. We use a parent pool size of 150 and a child pool size of 200, and run for 1500 generations.

4.3 Linear Programming Approach

The team formation problem can be formulated as an integer programming problem as follows:

$$\begin{aligned} \text{MINIMIZE} \quad & \sum_{i=1}^{n} c(a_i) \cdot x_i \\ \text{SUBJECT TO} \quad & \sum_{i:t_j \in \alpha(a_i)} x_i \geq \quad k+1 \quad j = 1, \ldots, m \\ & x_i \in \quad \{0,1\} \end{aligned}$$

The variable vector x represents our choice of including agents in the team; if $x_i = 1$, then agent $a_i \in \mathcal{A}$ belongs to our team, and otherwise they do not. In the LP relaxation of the integer program, we allow the integral vector x to assume real values with $0 \leq x_i \leq 1$. This is an exact solution to the fractional weighted set multicover; that is, if we could choose "fractions" of agents to solve some tasks, then this would be an optimal team. To compute the solution to the LP relaxation, we use the linear programming package PuLP [16]. Since algorithm times are not compared in this paper (since we implemented NSGA-II in a different language from the others) there was no difference in choosing PuLP over other state-of-the-art LP solvers.

To translate the LP relaxation into a team formation solution, we order agents by $a^{(1)}, \ldots, a^{(n)}$ such that $x^{(1)} \geq \cdots \geq x^{(n)}$. The team chosen is the smallest cutoff point c such that the team $\{a^{(1)}, \ldots, a^{(c)}\}$ is k-robust. However there is no guarantee that this ordering may provide an optimal solution. We are interested in finding an approximation bound on this method, and use a randomized rounding mechanism as part of our proof [18]. Consider the random algorithm for solving the team formation problem that performs the following during each iteration of the algorithm: For each agent a_i that does not yet belong to our team, place a_i in the team with probability x_i, where x is the solution vector to the linear relaxation of the integer program. Our method will perform about as well as this random algorithm, since it selects the most likely agents to be chosen in the random method.

Theorem 1. *The randomized rounding method is a* $2\ln(n+k) + O(\ln n)$*-factor approximation of the TORTF problem with probability* $\frac{(n-1)^k}{n^k}$.

Proof. Let OPT_f to be the optimal cost of the fractional set k-cover, and OPT to be the optimal cost of the integral set k-cover. Our proof will show that the random approximation is probably and approximately a factor of OPT_f, and since $OPT_f \leq OPT$, a factor of OPT as well.

Let x be the solution vector to the linear relaxation, and a corresponding objective value $\mathrm{c}(x)$. We will randomly construct a team in an iterative manner: for each iteration, we add agent a_i to our team with probability x_i. Consider the probability of a $t \in \mathcal{T}$ not being 1-robust after the first iteration, given that the associated probabilities of adding one of the p agents that can complete t is $x_1, \ldots, x_p$. This is the complement of having no items covered during that iteration:

$$P(t \text{ becomes 1-robust}) = 1 - (1 - x_1) \cdots (1 - x_p) \tag{3}$$

$$\geq e^{-x_1} \cdots e^{-x_p} = e^{-x_1 - \cdots - x_p} \geq e^{-k} \tag{4}$$

Since $x_1 + \cdots + x_p \geq k$ as a property of the linear program. Suppose q agents in our team cover t; let the probabilities of the uncovered agents being selected by $x_1, \ldots, x_{p-q}$ (we arbitrarily reorder x for simplicity). Clearly, since each $x_i \leq 1$, it follows that $x_1 + \cdots + x_{p-q} \geq k - q$, and the lower bound on the associated probability of having at least one item in this set cover t is e^{-k+q}.

The expected cost of performing an iteration in this algorithm, for $q = 0$, is $\sum_{i=1}^{n} E[c(a_i)x_i] = \sum_{i=1}^{n} c(a_i)x_i = OPT_f$, where OPT_f is the optimal cost of the fractional k-cover. For any $q > 0$, it should be clear that the cost per iteration is bounded above by OPT_f.

Now, suppose that we run this greedy method for $\frac{2}{k} \ln n$ rounds; then, the likelihood of not covering task t is $(e^{-k})^{2/k \ln n} = \frac{1}{n^2}$. The probability of any task being uncovered is $\frac{1}{n}$. Then, suppose we have q-robustness so far in the algorithm, and suppose it runs for $\frac{2}{k-q} \ln n$ iterations. The probability of not covering all tasks is $n(e^{-k+q})^{2/(k-q) \ln n} = \frac{1}{n}$. We repeat this for $q = 1, 2, \ldots, k$; the probability of having any tasks uncovered during any of these steps becomes

$$\prod_{q=0}^{k-1} \frac{n-1}{n} = \left(\frac{n-1}{n} \right)^k. \tag{5}$$

This algorithm runs in

$$\sum_{q=0}^{k-1} \frac{2}{k-q} \ln n = 2 \left(1 + \frac{1}{2} + \cdots + \frac{1}{k} \right) \ln n \tag{6}$$

$$= 2H_k \ln n. \tag{7}$$

Since $H_k = \ln k + O(1)$, the total number of iterations for this algorithm is approximately $2 \ln(n + k) + O(\ln n)$. Therefore, with probability $\frac{(n-1)^k}{n^k}$, this algorithm will approximately be a factor $2 \ln(n + k) + O(\ln n)$ of OPT_f.

The likelihood of the randomized rounding algorithm being within this factor is largest for large n and small k. These cases are more relevant to our interests, since we usually do not a huge robustness when forming teams, but we are interested in cases when dealing with a large number of candidate team members. □

5 Results and Discussion

To test our approximation algorithms, we compare their average cost performance on varying datasets. These datasets are constructed to reflect realistic scenarios in selecting a robust team. As we will show, each algorithm is able to perform marginally better than the others in different scenarios.

5.1 Datasets

We constructed several sample datasets to represent realistic team formation scenarios.

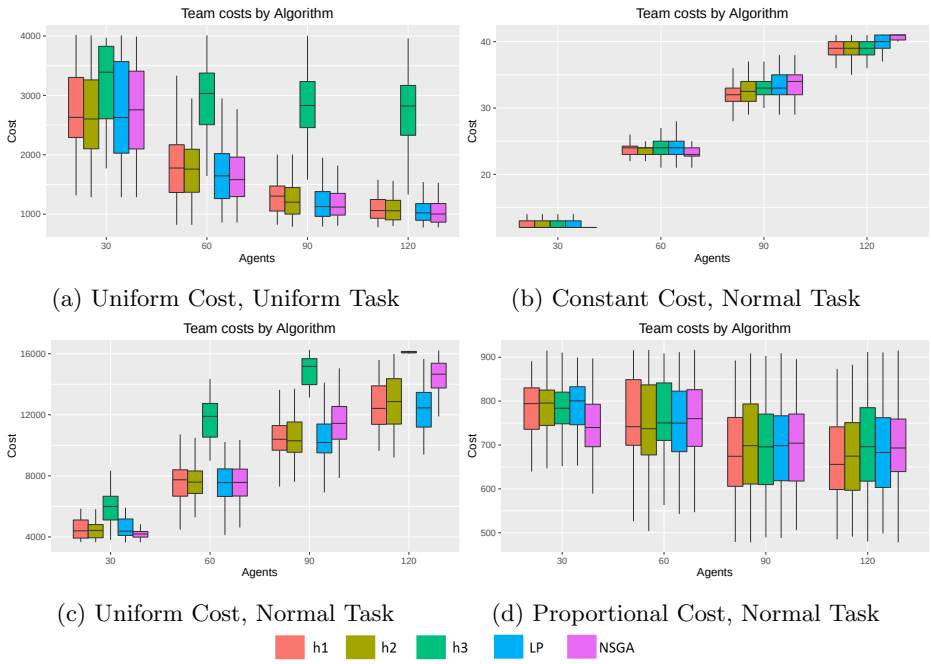
Uniform Cost, Uniform Tasks: Agent cost is uniformly distributed from 1 to 1000, and the number of tasks an agent can do are uniformly distributed from 1 to m. This dataset is similar to that used in [17].

We consider other datasets where the number of tasks are normally distributed. For the following three datasets, the number of tasks is normally distributed on $N(18\frac{m}{n}, 5)$. The mean was chosen because the systems using this value would have a maximal k-robustness of 5 to 14, depending on the configuration. A high maximal robustness guarantees that there are a large number of k-robust teams for sufficiently small k.

Constant Cost, Normal Tasks: The cost is kept constant at $c(a) = 1$. This is equivalent to the unweighted variant of the set multicover problem. The cost of hiring different team members may not be significantly different. Therefore, it is more important to minimize the size of the team than reduce cost.

Uniform Cost, Normal Tasks: In the cases where agents do carry a cost, their cost may not correlate with the number of tasks they can perform. In this dataset, cost is uniformly distributed from 1 to 1000. It considers the scenario in which each agent can perform approximately the same number of tasks, but their costs may vary widely. For example, a business hiring contract workers to complete a project will select from a pool of workers that can complete approximately the same number of tasks. However, the amount that each worker wants to be paid may not be a function of how many tasks they can complete, so there may be no correlation between the two.

Proportional Cost, Normal Task: The cost for each agent is determined by the number of tasks they can perform. Let $T = |\alpha(a)|$; the cost for each agent is sampled uniformly from $N(2 \cdot T, 2)$. This corresponds to many scenarios where agents are valued based on how many tasks they complete. For example, when forming a team of robots, the cost of adding a member would depend on what tasks they complete. If each task is equally challenging to complete, then the cost of an agent should be directly proportional to the number of tasks they can complete. The challenge of this problem is that forming teams is "approximately additive;" that is, the cost of hiring 4 agents to perform n tasks is approximately



(a) Uniform Cost, Uniform Task

(b) Constant Cost, Normal Task

(c) Uniform Cost, Normal Task

(d) Proportional Cost, Normal Task

Fig. 1: Distribution of costs per algorithm performed on 60 instances (for each number of agents) on multiple datasets with $m = 110$ and $k = 4$.

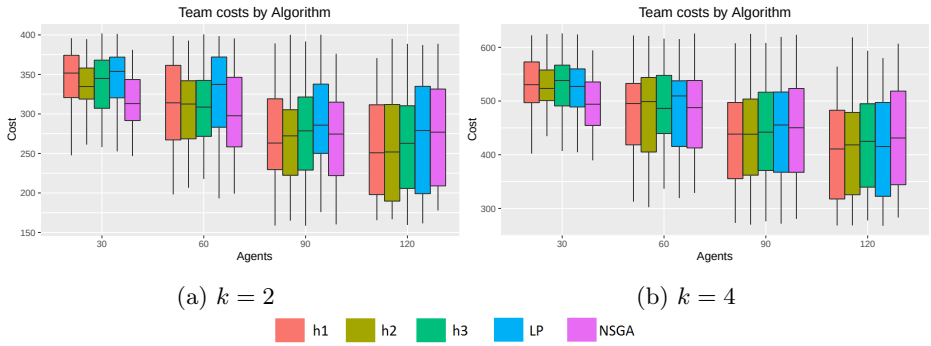
equal to the cost of hiring one agent to do those tasks. Therefore, teams with the same k-robustness should have approximately the same cost; since the difference in cost will be due mostly to noise, the algorithm will need to maximize cost per noise.

5.2 Results

We compare the algorithms across multiple metrics. Since we are not aware of any other state-of-the-art approximation algorithms for solving the TORTF problem or set multicover, only the algorithms here are compared. Figure 1 compares the average costs of teams computed by each algorithm for each dataset and a varying numbers of agents. The cost of teams usually increases with the number of agents available since, in the three normally distributed datasets, the number of tasks assigned to each agent is inversely proportional to the number of agents in the system. When the number of tasks is uniformly distributed, the average cost decreases, as is expected. Surprisingly, the performance of the heuristic algorithms for different k values is not significantly different, as shown in Figure 2. One point to note is that the variance in performance of the algorithms decrease, i.e., they perform more consistently when for larger values of k. This is because the number of at least k-robust teams decreases as the value of k

Algorithm	Avg. Error	% Correct
Heuristic 1	7.12	21.67
Heuristic 2	7.05	26.67
Heuristic 3	4.36	28.33
Linear Program	4.76	**36.67**
NSGA-II	**0.03**	5.00

Table 1: Algorithm cost performance against the optimal solution, computed on 60 instances of the proportional cost/normal task dataset with 30 agents, 20 tasks and $k = 2$.



(a) $k = 2$ (b) $k = 4$

Fig. 2: Distribution of costs per algorithm performed on 60 instances (for each number of agents) of the proportional cost/normal task dataset for varying k.

increases. Table 1 compares our algorithms' performances against that of an integer program solver [16] (which we were able to run only on very small problem sizes), on the proportional cost/normal task dataset for few agents and tasks. The optimal solutions were computed using an integer program solver in the PuLP package. Average error is the average difference between the algorithmic solution; percent correct is the percentage of solutions that achieved optimal cost. This comparison measures the approximation quality of the algorithms for smaller problem sizes. While the linear program and heuristic algorithms achieve the highest percentage of optimal teams, NSGA-II consistently finds teams with near-optimal cost.

Greedy Heuristic 1 selects the agent with minimum cost per task. As figure 1 shows, Heuristic 1 performs well in all datasets and over all agent sizes. The greedy decisions made by Heuristic 1 make it likely to choose some optimal agents, and it has the strictest known approximation bound on the heuristics we compare. For smaller number of agents, its performance was not optimal, but it outperforms all other algorithms for larger agent sizes. In particular, Heuristic 1 generates the most cost-effective team on average when there are a large number of agents, since it is guaranteed to generate at most a H_n-factor approximation. In comparison to NSGA-II, Heuristic 1 found optimal-cost teams at a much higher rate. This is due to the construction of the proportional cost dataset and

the fact that the heuristics we implemented choose agents with minimum cost per task. For example, if costs were exactly equal to the number of tasks that an agent can perform, then Heuristic 1 will always assign certain optimal agents the minimum price. Let $O \subseteq \mathcal{A}$ be the optimal team, and let C be the set of tasks that are not k-covered by the current greedy team G. If there is an agent $a^* \in O$ such that $\alpha(a^*) \subseteq C$, then its cost will be $c(a^*)/|\alpha(a^*)| = 1$, the minimum cost possible. Therefore, the greedy algorithm will always have a preference towards choosing optimal teams.

Like Heuristic 1, Heuristic 2 performs moderately well on all datasets. For a smaller number of agents, heuristic 2 performs better than heuristic 1. In cases with constant or proportional cost, Heuristic 3 performs competitively with the other greedy heuristics. Among the heuristics and linear program performances, Heuristic 3 has the minimum cost difference with the optimal solution 1. On datasets using uniform cost, Heuristic 3 vastly underperforms the other algorithms, which suggests that marginal utility calculations fail to work when agent costs have high variance or are uncorrelated with the number of tasks the agent performs.

The linear program approach that we use is the most consistent in its performance; when comparing it against the other algorithms presented, it often performs among the top 3. Table 1 also shows that it can often find optimal solutions in a dataset where the optimal team has many near-optimal teams. Since the approximation factor on the linear program is similar to that of the greedy algorithm, it is not surprising that their performances are also similar.

The NSGA-II algorithm performed best with a smaller number of agents across all datasets, as indicated in Figure 1. We observed that its relative performance was not affected by the number of tasks. Since the size of the search space for the GA increases exponentially with the number of agents in the problem increasing, this is not a surprising trend. Moreover, since GAs search locally about existing solutions to make improvements, NSGA-II often finds near-optimal results, as shown in Table 1. Since the NSGA-II algorithm computes the entire Pareto frontier of solutions, it runs much slower than the greedy approximation and the linear program. Therefore, NSGA-II does best in scenarios with fewer agents or when multiple values of k need to be computed.

6 Conclusions

We present and experimentally compare a number of approximation algorithms for producing efficient k-robust team for the team formation problem. Our heuristic methods are based on a naive greedy algorithm that is proven to be a H_n-factor of the optimal solution. We also present a linear programming relaxation of the optimal, but costly, integer programming formulation, which has approximately the same factor approximation as Heuristic 1. These methods are compared to a NSGA-II based genetic algorithm approach that approximates the Pareto frontier of solutions to the team formation problem instead of finding an optimal team for one k.

Results indicate that the NSGA-II method performs near-optimally for lower number of agents, but the heuristic and linear programming mechanisms scale up better in terms of cost-effectiveness. The NSGA-II method takes much longer to run compared to the other heuristics, so it is best to use with a large number of tasks or when multiple robustness levels need to be compared. Heuristic and linear programming techniques are a better fit for large populations.

The weighted set multi-cover problem, which has been studied extensively in literature, has been applied to many different types of problems. To study more realistic scenarios, we may want to examine cases where an agent can only complete a fraction of a task – multiple agents would be needed to complete a single task (that is, achieve 1-robustness for a task). Another possible direction would be to restrict the number of tasks that an agent can be assigned to complete. This would map to real-life scenarios where team members may be capable of completing many tasks, but time and/or resource constraints restrict them from completing all of them.

References

1. Y. Bachrach, P. Kohli, V. Kolmogorov, and M. Zadimoghaddam. Optimal coalition structure generation in cooperative graph games. In *AAAI*, 2013.
2. Y. Bachrach and J. S. Rosenschein. Coalitional skill games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 1023–1030. International Foundation for Autonomous Agents and Multiagent Systems, 2008.
3. S.-J. Chen and L. Lin. Modeling team member characteristics for the formation of a multifunctional team in concurrent engineering. *Engineering Management, IEEE Transactions on*, 51(2):111–124, May 2004.
4. V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.
5. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
6. U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.
7. Q.-S. Hua, Y. Wang, D. Yu, and F. C. Lau. Set multi-covering via inclusion–exclusion. *Theoretical Computer Science*, 410(38):3882–3892, 2009.
8. Q.-S. Hua, Y. Wang, D. Yu, and F. C. Lau. Dynamic programming based algorithms for set multicover and multiset multicover problems. *Theoretical Computer Science*, 411(26):2467–2474, 2010.
9. Q.-S. Hua, D. Yu, F. C. Lau, and Y. Wang. Exact algorithms for set multicover and multiset multicover problems. In *Algorithms and Computation*, pages 34–44. Springer, 2009.
10. D. S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49. ACM, 1973.
11. R. M. Karp. Reducibility among combinatorial problems. In R. Miller, J. Thatcher, and J. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

12. H. Kitano. Robocup rescue: a grand challenge for multi-agent systems. In *MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, pages 5–12, 2000.
13. T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 467–476, New York, NY, USA, 2009. ACM.
14. L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete mathematics*, 13(4):383–390, 1975.
15. L. S. Marcolino, A. X. Jiang, and M. Tambe. Multi-agent team formation: diversity beats strength? In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 279–285. AAAI Press, 2013.
16. S. Mitchell, M. OSullivan, and I. Dunning. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand, http://www. optimization-online. org/DB_FILE/2011/09/3178. pdf*, 2011.
17. T. Okimoto, N. Schwind, M. Clement, T. Ribeiro, K. Inoue, and P. Marquis. How to form a task-oriented robust team. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 395–403. International Foundation for Autonomous Agents and Multiagent Systems, 2015.
18. P. Raghavan and C. D. Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.
19. O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(12):165 – 200, 1998.
20. V. V. Vazirani. *Approximation Algorithms*. Springer Science & Business Media, 2013.
21. H. Wi, S. Oh, J. Mun, and M. Jung. A team formation model based on knowledge and collaboration. *Expert Systems with Applications*, 36(5):9121 – 9134, 2009.