Exploiting spatial locality and heterogeneity of agents for search and rescue teamwork

James Parker, Ernesto Nunes, Julio Godoy, and Maria Gini
Department of Computer Science and Engineering,
University of Minnesota, 200 Union St SE, Minneapolis, MN 55455, USA
{jparker,enunes,godoy,gini}@cs.umn.edu

Abstract

We propose coordination mechanisms for multiple heterogeneous physical agents that operate in city-scale disaster scenarios, where they need to find and rescue people and extinguish fires. Large scale disasters are characterized by limited and unreliable communications, dangerous events that may disable agents, uncertainty about the location, duration and type of tasks, and stringent temporal constraints on task completion times. In our approach, agents form teams with other agents that are in the same geographical area. Our algorithms either yield stable teams that are formed upfront and never change, or fluid teams where agents can change teams as need arises, or teams that restrict the types of agents that can belong to the same team. We compare our teaming algorithms against a baseline algorithm in which agents operate independently of others, and two state-of-the-art coordination mechanisms. Our algorithms are tested in city-scale disaster simulations using the RoboCup Rescue simulator. Our experiments with different city maps show that, in general, forming teams leads to increased task completion and, specifically, that our teaming method that restricts the types of agents in a team outperforms the other methods.

1 Introduction

The massive western Colorado mudslides, hurricane Irene and the earthquake and tsunami in Japan are just a few examples of large scale disasters that occurred in recent times. Recent developments in AI and robotics have produced multi-robot systems that can be used to help mitigate the damage from these disasters. Robots have been used in several post-disaster scenarios in urban areas (Kruijff et al., 2012; Murphy, 2014) and in the Fukushima disaster (Kawatsuma et al., 2012). Stringent temporal constraints and limited situational awareness in disasters require efficient multi-agent task allocation methods and the ability to plan under uncertainty.

In these types of environments, there is uncertainty as to the location, type and size of tasks, and agents must travel through possibly unsafe areas to reach the tasks. The environment is dynamic, fires will spread, buildings collapse, and civilians die if they are not rescued promptly. All these elements make urban search and rescue a very demanding testbed and, at the same time, a realistic and important application of AI techniques.

The ultimate goal in urban search and rescue is to contain the damage, namely, to maximize the number of lives saved while minimizing their injuries, and maximize the amount of infrastructure preserved while minimizing the damage to these infrastructures. The algorithms we propose have these goals as their optimization objectives, and we measure their performance with respect to those objectives.

Our research hypothesis is that if agents work as part of teams their performance will increase because they will be able to take advantage of synergies among them. We define a team to be a set of agents who work in the same spatial area and typically remain together for extended periods of time. Teams of agents are initially formed using hierarchical clustering, which creates teams that are spatially compact and have a similar number of agents. To maintain the spatial locality of the team, agents on a team select tasks that are close by by reducing the value of tasks that are distant. Sometimes it is useful to transfer agents to a different team, for instance when more tasks need to be completed in the area covered by another team. Restricting team membership to specific types of agents can also, in same cases, avoid having conflicting goals in a team and hence improve performance.

Our research objective is to answer experimentally a number of research questions on the performance of agents in highly dynamic search and rescue environments. We specifically address the following questions: (1) In highly dynamic environments, should non-greedy task allocation methods be preferred to greedy ones? (2) Does forming teams help improve performance? (3) Does transferring agents across teams improve performance? (4) Lastly, does taking agent type synergies into account when forming teams help improve performance?

To achieve our objectives, we have chosen to conduct our experimental work using the RoboCup Rescue simulator for the agent competition (Kitano and Tadokoro, 2001), which has been developed over the years by the international research community to advance research in urban search and rescue and to provide evaluation benchmarks for rescue strategies. The simulator models a world that is suitable for autonomous agents that have limited functionalities, sensors, and communication capabilities (Murphy et al., 2008). This is not exactly what happens on the ground, yet the simulator is a valuable research tool.

On the ground, human rescue teams work within established organizational structures and follow well-defined guidelines from emergency rescue agencies, such as the Federal Emergency Management Agency (FEMA) in the U.S (Waugh and Streib, 2006). Communication is not usually an issue, since one of the first tasks of rescue teams is to establish communication links between command control centers and search and rescue responders (Cooper, 2005). When robots are used in rescue field operations, they are not autonomous (Kruijff et al., 2012). That notwithstanding, the RoboCup Rescue simulator provides many benefits, such as the availability of an open source tool to study the effect of different ways of making decisions and different coordination methods in large-scale maps of real cities. Being able to assess the impact of the lack or scarcity of communications on rescue operations can help design more robust decision making methods. Since the simulator, maps, and competition results are available, anyone can compare results obtained using different algorithms.

Our results are obtained only in simulation. This is because the scale of the city maps where teams operate is such that today it is not possible to use real robots at the same scale to validate the simulation results. Testing deployment and rescue strategies in real-world training exercises is time consuming and expensive, making it impossible to do the rigorous experiments needed to obtain a detailed understanding of the interactions between the environment and the strategies of the rescuers. Simulations can be run on an almost endless number of situations, data can be collected and analyzed to refine strategies. It is also relevant to note that studies on the effectiveness of training first responders and public safety personnel to respond to disasters using simulators showed improved performance when compared to traditional field exercises (Kincaid et al., 2003).

This work makes several contributions:

1. We propose distributed team formation algorithms for environments where (1) tasks are not all known upfront, but have to be discovered and can appear and disappear dynamically; (2) there is significant uncertainty on where the tasks and the other agents are; and (3) communication between agents is highly constrained and subject to transmission errors. Most of these features are found in real-world large-scale rescue situations.
2. We offer domain specific and computationally frugal task selection heuristics that allow individual

agents to quickly decide what tasks to tackle first, and can handle agent failures and discovery of new tasks.

3. We evaluate the effectiveness of our decision making algorithms and compare them with state of the art methods using the RoboCup Rescue simulator as our evaluation testbed. The use of the RoboCup Rescue simulator, with its standard settings and the maps used in the 2013 competition, makes our results easy to replicate and comparable with others. This fills in a gap in the current literature, where only a handful of the papers use the full simulator settings.

The methods we propose are applicable to a broad set of domains outside search and rescue, such as military surveillance and reconnaissance, that are characterized by many agents and tasks, complex and dynamic environments, limited time to do the tasks, and uncertainty.

In the rest of the paper, we give an overview of the RoboCup Rescue simulator in Section 2 and we compare various approaches to task allocation and teamwork from the literature in Section 3. We formalize the team formation problem describe how agents evaluate and select tasks when not in a team in Section 5. Section 6 describes how agents select tasks when on teams and how teams are initially formed. The different team formation models are described in Section 7. Section 8 describes our experimental setup and parameter settings. We present experimental results in Section 9 where we compare the strengths and weaknesses of the different team configurations for a wide variety of scenarios. In Section 10 we evaluate our results in the context of the RoboCup Rescue competition. Lastly, in Section 11 we offer concluding remarks and ideas for future work.

2 Background on RoboCup Rescue

The RoboCup Rescue simulator is the result of a continuous effort within the AI community to create a framework that simulates a realistic post-disaster scenario in a city (Kitano and Tadokoro, 2001) and to evaluate the effectiveness of different strategies. The framework allows researchers to design multi-agent coordination methods to save civilians and reduce fire damage to the city. This simulator is part of the RoboCup Rescue Simulation Agent League, an official event at the annual RoboCup competition (Akin et al., 2013).

In the simulator, agents play different roles defined by the actions they can perform:

- Ambulances search for wounded civilians, rescue them and take them to a refuge.
- Fire trucks look for fires and put them out, refilling their water tanks as needed.
- Police clear debris which blocks roads to enable the other agents to move through the city and accomplish their tasks.

Besides the agents, the simulation scenario is composed of buildings, which may be in good condition, on fire, or totally burned; roads, that may be open or have various amounts of blockages; central police offices, central fire stations, and ambulance centers; and refuges, where civilians go for safety and where the fire trucks can refill their water tanks. Fire hydrants are also scattered around the city to enable fire trucks to refill water.

Figure 1, shown later in Section 9.1 is a screenshot from the simulator. The blue, white, and red circles on the maps represent respectively the police, ambulance, and fire truck agents. The bright green, dull green, and black circles represent healthy, wounded, and dead civilians respectively. The yellow, orange, and maroon colors represent the increasing intensity of fire in buildings; black represents completely burned and destroyed buildings. There are two special types of buildings: the red buildings with a house icon are refuges where saved civilians are taken; the police hat, garage building, and medical cross are centers for police, fire trucks and ambulance, respectively. Fire hydrants are represented by the red hydrants on the light gray color roads. The black polygons on the roads are the impassable blockages.

The simulation has many constraints, which have been designed to emulate a post-disaster scenario. The agents have a map of the city, but they do not know where street blockages are and where the tasks are located. They can detect only the tasks close to them. Civilians have a limited life span while trapped under building rubble and will perish if they are not rescued or the building they are trapped under catches fire. Centers, like any other building, can burn down and cease to function. Agents entering a burning building will become wounded, and these wounds will eventually result in their death unless they receive first aid treatment at a refuge.

Agents can share information about their discoveries and task assignments with other agents through messages that are sent via either long-range radio or short-range shouts. Long-range communication is limited by a fixed bandwidth and a limit on the number of channels an agent can subscribe to, and is susceptible to errors and message corruption. Centers typically can subscribe to more channels than agents but cannot perform tasks.

The simulator itself has evolved from its inception in 1999, with contributions from many researchers across the world (e.g., (Farinelli et al., 2003; Akin et al., 2013)). Changes made in 2009 (Skinner and Ramchurn, 2010) have produced a completely new implementation, all written in Java and built around polygonal elements. The method used to compute the score in the RoboCup Rescue competition has also been changed (Siddhartha, 2009). A converter from publicly available maps enables users to import real cities maps into the simulator (Göbelbecker and Dornhege, 2009).

The RoboCup Rescue simulator includes, by default, a set of "sample" agents for each agent type. These agents do not communicate but simply try to accomplish tasks as they find them: police clear blockages, fire trucks put out fires in buildings, and ambulances rescue civilians. If an agent has not encountered any task of the type it is capable of doing, then it will randomly walk around until it finds a suitable one. While the sample agents are very basic and greedy, they provide a standardized baseline for testing.

3 Related Work

Allocation of resources and tasks in multi-agent systems and the use of teamwork to improve performance are well-studied topics. A survey of the issues with a categorization of task allocation problems is provided in (Chevaleyre et al., 2006). Following their categorization, our domain of interest focuses on discrete tasks, which are not divisible but sharable, are non stable, and are additive. In addition, our work uses utility functions that are utilitarian (i.e., the sum of individual utilities).

3.1 Task Allocation

Multi-agent task allocation is the problem of matching each task in a set K of tasks to at least one agent in a set A of agents, in scenarios where usually $|K| > |A|$, while trying to maximize the agents' rewards (or minimize their costs). The problem of finding an optimal task allocation is NP-complete and is inapproximable (An and Lesser, 2010), even when restricted to situations where each agent has complete information about the other agents. The general problem when agents do not have full observability is NEXP-complete (Goldman and Zilberstein, 2004).

In RoboCup Rescue all the tasks can be done by a single agent but some tasks are completed faster when multiple agents work concurrently on them. For instance, a fire can be extinguished faster if multiple fire trucks work on it, a civilian can be unburied faster by multiple ambulances, but only one ambulance is needed to take the civilian to a refuge.

Approaches for multi-agent task allocation can be broadly classified into centralized and decentralized. Centralized methods assume a central entity that computes the allocations and assigns the tasks to agents. Many approaches to task allocation are centralized. The assignment of tasks that require a single agent to agents that do only one task is called a linear assignment problem and can be solved with the Hungarian algorithm (Kuhn, 1955). Many algorithms have been developed for more complex cases, such as tasks that require multiple agents, agents that can do multiple tasks at the same time, tasks with precedence constraints, and more (Korsah et al., 2013). Exact centralized methods produce optimal solutions, but have poor scalability (Bertsimas and Weismantel, 2005) and are not robust to failure since they rely on a single entity to collect information and make decisions.

In RoboCup Rescue the environment is dynamic, not all the information is known upfront, and information may become stale because of communication delays, task expiration or robot failures. In such situations, a centralized system requires high-bandwidth to share information, which is problematic. For these reasons, decentralized methods are much more common in RoboCup Rescue.

Decentralized task allocation is often modeled as a distributed constraint optimization problem (DCOP) (Shoham and Leyton-Brown, 2009) and solved using DCOP methods. One such method is the approximate max-sum algorithm (Farinelli et al., 2008) which has been used for task allocation in sensor networks and in RoboCup Rescue (Ramchurn et al., 2010a). We use the max-sum algorithm as one of the benchmark algorithms to which we compare the performance of our team formation algorithms (see Section 9.4). LA-DCOP (Scerri et al., 2005) is an approximation algorithm that uses token passing (Xu et al., 2005; Farinelli et al., 2006) as follows: When an agent perceives a task, it creates a token to represent it. It can decide to do the task or pass the token to a randomly chosen agent. This tends to guide the search quickly towards a solution, but the algorithm is greedy. (Ferreira et al., 2010) compared LA-DCOP with their method, Swarm-GAP, in which an agent chooses a task according to a probability that depends on the stimulus generated by the task and the agent's threshold. Results show that both DCOP approaches behave similarly, and both perform better than a greedy task allocation. Their experimental work uses RoboCup Rescue, but does not attempt to coordinate the different types of agents as our work does.

Recently, to facilitate comparing the performance of DCOP algorithms, RMASBench, a system that provides a library of state-of-art solvers for DCOP and for comparing them, has been created and added to the annual competition (Kleiner et al., 2013). Two common shortcomings of DCOP-based methods in RoboCup is that they do not take full advantage of inter-type synergies among agents (Ferreira et al., 2010), and their communication requirements violate the communication constraints in RoboCup Rescue (Pujol-Gonzalez et al., 2014).

Auctions are another widely used method for making decentralized decisions. Auctions require communication between the auctioneer and the agents, which might be problematic. However, complete communication requirements can be avoided using consensus based methods (e.g., (Zavlanos et al., 2008; Choi et al., 2009)), where each robot determines independently its tasks, and an equilibrium is reached by iteratively sharing information with the local neighbors.

Auctions have been used in RoboCup Rescue with limited success because of their communication requirements. For instance, (Sedaghat et al., 2006) proposes an auction-based mechanism in which all non-police type of agents make requests for clearing blockages to their respective centers which forward the requests to the police center (auctioneer). The police center notifies police agents of the tasks, police agents send their bids, and the center assign each task to the most adequate police agent. This requires a large number of messages, which can easily exceed what is allowed. To drastically reduce the communication needs during the bidding and allocation phases of the auctions, in (Nanjanath et al., 2010) each center uses proxies for the agents of its own type. We use this approach as one of our benchmark algorithms.

Other decentralized task allocation methods nonspecific to RoboCup include methods that partition the problem space and allocate parts of it to each agent to reduce the computational complexity and limit the need for coordination (Matignon et al., 2012; Liu and Shell, 2012b), allocate tasks via swaps among neighboring agents (Zheng and Koenig, 2009; Liu and Shell, 2012a), or allocate tasks greedily according to agents' capabilities and availability (Gunn and Anderson, 2013). While the methods that rely on longer-term planning could be made ineffective by the fast changing nature of search and rescue, the method proposed in (Gunn and Anderson, 2013) could be extended to RoboCup. In fact, their task allocation is similar to ours in that both methods are greedy. However, unlike their method in which allocation is done by a team leader, in ours agents choose their tasks independently. This may cause more than one agent to be allocated to the same task, but this is acceptable in RoboCup where a task is completed faster if multiple agents work on it together.

The work in (Trăchioiu, 2014) breaks down the problem into a macro and micro subproblems similar to how we create teams on a macro level and select individual tasks on a micro level. They compare a wide variety of approaches in literature, but focus on subproblems where a single type of agent is present. Their method did not score very well in the RoboCup competition. This indicates that optimizing the algorithms that control each agent type is not sufficient in RoboCup Rescue. Our methods include agents of multiple types in the teams to enable implicit coordination between agents and are able to score much higher.

3.2 Teamwork and Coordination

To increase the efficiency at completing tasks, agents can work together in coalitions or teams. We use the term *teams* to convey more clearly the cooperative nature of agents in our domain, but the term *coalitions* is also common in the literature.

Forming an optimal coalition of agents for a set of tasks requires an amount of computation exponential in the number of agents N (the coalition search space is 2^N). Brute-force algorithms that search the entire space of possible coalitions are unsuitable, even for offline algorithms. Anytime algorithms were proposed in (Shehory and Kraus, 1998) for forming coalitions for different types of tasks and with either disjoint or overlapping coalitions. For the general case of agent coalition formation, (Sandholm et al., 1999) proposed an approximate algorithm that produces a solution within a bound from the optimal. (Dang and Jennings, 2004) and (Manisterski et al., 2006) improved Sandholm's algorithm by further reducing the search time, which, however, remains exponential in the number of tasks. Even with those improvements, the complexity of coalition formation makes the approaches impractical in real-time situations.

Other team formation algorithms include, for instance, a graph-based algorithm (Gaston and des Jardins, 2005) where an agent either forms a new team when there is an unassigned task or joins an existing team. An agent moves to another team when its performance is higher than the average performance of a neighboring team. Another graph-based approach (Liemhetcharat and Veloso, 2012) models synergies among agents in a heterogeneous multi-agent system by building a synergy graph from which teams are formed. Once formed, the teams cannot be changed. This method is not appropriate for RoboCup Rescue because the time necessary to create the synergy graph will reduce the time available to do the tasks.

Coalition and team formation algorithms used for agents are not directly applicable to robots, as real-world environments introduce extra constraints to the problem, such as physical distance between robots and tasks, obstacles in the environment, and limited power supply for the robots. Some approaches (e.g., (Vig and Adams, 2007; Service and Adams, 2011; Zhang and Parker, 2013)) address specifically these differences.

For example, (Zheng and Koenig, 2008) coordinate coalitions of robots for tasks requiring more than one agent in order to minimize either the waiting time (when all robots must be present at the task in order to execute it) or the total path cost of all the robots. The approach is limited by the complexity of coordinating many agents' schedules. Instead, they rely on an approximate solution. However, their approach assumes that agents are homogeneous, that tasks are known beforehand and that robots can follow direct straight paths to the tasks, while in the RoboCup Rescue domain there are different types of agents, tasks need to be discovered and paths leading to them might be blocked by obstacles.

In (Vig and Adams, 2007) robots have different roles depending on their capabilities and on their knowledge of the tasks. When an agent finds a task that it cannot accomplish alone or is given information about such a task, it becomes that task's "proxy". In this approach, the bidding process is reversed, with tasks bidding for agents. The approach is useful in cases where tasks are not known upfront, because it avoids the expensive computations required to find the best coalition. However, this method assumes communication between the robots. In RoboCup Rescue communication is limited and error prone. Complexity and inapproximability results are provided in (Service and Adams, 2011).

To reduce the complexity of evaluating all possible coalitions, (Zhang and Parker, 2012) consider only executable coalitions, i.e. coalitions for which the preconditions are satisfied. Sensing limitations in the robots can reduce significantly the number of executable coalitions compared to the number of feasible coalitions, making the problem more tractable in practice. In our work, we also reduce the coalition search space by considering the spatial locality of the agents. (Zhang and Parker, 2013) also consider inter-task constraints and proposes some greedy heuristics that have solution guarantees.

(Ramchurn et al., 2010a) describe a DCOP formulation for the coalition formation problem that is solved using the F-Max-Sum algorithm to allocate tasks with spatial and temporal constraints to coalitions. In (Ramchurn et al., 2010b) a 'look-ahead' heuristic is proposed which allocates the smallest possible coalition to a task. The rationale behind this choice is to maximize the amount of time agents work on tasks and minimize the amount of time they move between tasks. Furthermore, to avoid greedy assignments, the algorithm uses a one-step look ahead when assigning coalitions (thus the name) to maximize the number of future tasks available to each coalition. Although the authors refer to the RoboCup Rescue platform as a motivating application and potential domain, they do not use the RoboCup Rescue simulator and consider only ambulances and fire brigades. Instead, our approach involves teams formed with all types of agents, and we evaluate our approach experimentally with the RoboCup Rescue simulator.

Like (Vig and Adams, 2007; Zheng and Koenig, 2008; Zhang and Parker, 2012), our work involves teams of heterogeneous agents. We use the spatial location of the agents as a replacement for the synergy graph (Liemhetcharat and Veloso, 2012), and continuously assess team utility so agents can move to a different team whenever the change improves overall performance. Specifically, we propose three team formation mechanisms that differ for the conditions under which the agents can change their team membership.

4 Problem Definition and Assumptions

We formulate the team formation problem in dynamic environments as follows. Assume a finite set of agents A and a finite set of tasks K. Agents do not know where the tasks are and what they are, they need to discover them over time.

An agent $a \in A$ has a position, a type p, a set of actions, a set of observations, and a value for each task k, which we call Val_k. We assume that the position is a point in 2D, (x, y). In our case, the agent type is $p \in \{$*police, ambulance, fire truck*$\}$, but the algorithms are general and work with any number of types. For our work we assume that an agent can have only one type, but the framework can be extended if agents can perform multiple roles. In the case of RoboCup Rescue, each agent type can only perform one type of task. In general, agent types can be thought of as different goals. For example, fire trucks have the goal of extinguishing fires. What is important for this work is that some goals may be synergistic while others are conflicting. We also assume some goals interact with each other, either as a positive or negative effect, and that goals cannot be achieved optimally in isolation of considering the other goals. Section 7.3 shows how inefficiency can be reduced by forming teams without conflicting goals.

Agents can do different types of tasks depending on the agent type, but their actions are broadly defined as $\{$*doing a task, moving to a task, searching for a task*$\}$. Agents have a limited sensing range, and can observe the state of the environment, for instance the presence of other agents, road blockages, civilians trapped, or fires, only within that range. Agents can suffer damage from burning and die when their health is reduced to zero. The set of agents is fixed at the beginning; no more agents are added during the simulation.

A task $k \in K$ also has an (x, y) position, a type, and attributes that depend on the type of the task. In our case the task types are $\{$*blockage, fire, trapped civilian*$\}$. For example, a fire has a fieriness value, a civilian has a value for how deep it is buried and a value for its current health. The attributes of some tasks maintain the same value until an agent operates on them (e.g., road blockages), other change over time (e.g., the fieriness of a fire and the health of a civilian). Time is measured in discrete small time units and is indicated by t. Most tasks exist from the beginning but tasks can appear at any time during the simulation. Civilians are trapped from the beginning, but fires can start at any time.

We assume that at any time step an agent can compute a value for each task, Val_k (see Section 5 for details on how to compute it). Val_k is 0 if the agent cannot perform that task. We assume that an individual agent's ability is not affected by what other agents do. For example, two ambulances dig out a civilian at twice the speed a single ambulance does. If a single agent cannot dig out a civilian before she dies, then having more than one agent does more than speeding up the task execution, it enables task completion.

We assume that the agents are cooperative and share the same goal, which in our case is to save civilians and buildings. Agents attempt to do tasks to the best of their capability and report information correctly. Risks associated with doing tasks are not directly addressed in this paper, but they can be incorporated into the value a task, Val_k, to penalize risky tasks.

Agents can form teams $M = \{m_1, ..., m_j, ...\}$ from subsets of agents. A team $m \in M$ is a partition of the set of agents, namely $m \subseteq A$. An agent can belong to one and only one team, and a team can have agents of different types. Team formation is NP-complete even when all tasks are known upfront (Ramchurn et al., 2010b). Our problem is at least as hard and therefore is also NP-complete.

As mentioned before, when an agent discovers a task k, it computes a value for the task Val_k, which depends on the agent's distance to the task and the task status. Val_k changes over time as internal properties of the task change and agents move. When agents work in teams, each task has also a value for the team m, $TeamVal_{k,m}$, which is used by an agent to decide which task to do. This implicitly produces cooperation among the agents to complete shared tasks, where their added capabilities reduce the time to complete them. Details on the task value and the team value are in Section 5 and Section 6, respectively.

Agents communicate using messages of different types, through channels of limited bandwidth, to which they need to subscribe, as described in Section 2. Any message sent above the bandwidth is dropped without any notification to the sender. Two types of errors can occur when messages are sent on a channel: a transmission error, in which the message is not transmitted at all, and a receiving error, in which the message is sent correctly but it is received with no content. Each agent subscribed to a channel has a chance of getting a receiving error, but the sender is unaware if any receiving error occurred. The centers can subscribe to multiple long range channels simultaneously, and can be used to collect information from agents or teams and share information with them.

5 Working without Teams

In this section, we outline how each agent type evaluates tasks when there are no teams (called "base"). The task evaluation functions we describe use task specific information that an agent can observe. For instance, it would be useful to know the health of a civilian to determine whether an agent can rescue the civilian before she perishes, but that is not directly observable. We have tested empirically various ways of measuring the value of tasks, including an evaluation strictly based on distance. The methods we present are the ones that performed the best.

Accomplishing tasks quickly is critical because the value of tasks changes over time and new tasks are constantly being discovered. For example, if an ambulance does not rescue a civilian fast enough, the building might catch fire or the civilian might die by being crushed. At each time step, an agent needs to choose a task that is both relevant and that can be accomplished quickly.

Each agent evaluates tasks independently, and it greedily selects the task that has most value. More sophisticated planning approaches are possible, but this would require more computation power, more accurate knowledge for each agent, and the ability to handle tasks that appear or disappear. Ideally this planning should happen among all agents on a team but it would require more communication bandwidth than what is available on some maps. The greedy selection reduces on-board computation requirements, and enables the agent to react quickly to changes in the environment. Even when there is sufficient communication, the low observability of the environment makes these greedy heuristics outperform long-term planning, as shown in Section 9

Each agent also keeps track of all observations gathered throughout the entire simulation, and can share observations with other agents. For example, when an agent discovers a building on fire, it will send a message to the fire trucks with the location of the burning building. All fire trucks who are able to hear the message will add that burning building to their own local information. The information agents receive from other agents is not necessarily complete, since some messages are not delivered and not all the information can be shared due to communication limits, but it is nonetheless useful.

5.1 Task Evaluation for Fire Trucks

Burning buildings are tasks that require fire trucks. In order to accomplish task k as quickly as possible, an agent needs to minimize both the distance D_k to the task and the fieriness F_k of the fire. We define the value of a task for a fire truck as

$$Val_k = \frac{\alpha}{D_k \times F_k}.$$

Val_k is time dependent. However, we do not explicitly include time, because the agent computes the value in each decision cycle and decides what to do. In the next cycle, the agent will likely be in a different location and the fieriness of the fire might also be different. Some of this information is obtained via observations or messages from other agents, but some information is old and recalled from memory, and may no longer be accurate.

Since the scale of distance values is much larger than for fieriness, distance has a bigger impact on the decay of the value. The scaling factor, α, is experimentally set to $\alpha = 100$ for all the value calculations in this paper. This value keeps Val_k positive for all agent types and on the same order of magnitude. Buildings that are not on fire are not assigned a value and are not considered as possible tasks. Fires that have grown large have a large fieriness and are much more difficult to put out. Thus, by having fieriness in the denominator fire trucks will attempt first to put out close fires that are small.

Once the most valuable task is identified, the fire truck will head towards it and as soon as the building is in sight it will start extinguishing the fire. Fire trucks constantly look for better tasks. Hence, if a more valuable task appears, the fire truck will switch to it, even if the previous fire has not been put out yet. Take for example the situation of a fire spreading to a nearby building. Fire trucks will quickly put out the new small fire before returning to the old one to ensure the fire remains contained. This behavior is desirable for fighting fires since it is much easier to contain a fire than extinguish it completely. However, if a more committed behavior is desirable, Val_k can be biased to be much higher for the task currently being worked upon. When its water tank is empty, a fire truck goes immediately to a refuge or fire hydrant to refill its tank.

The fieriness is the most desirable property to include in the evaluation function, as it combines both the temperature and the rate at which the combustible material in the building burns. Temperature alone is insufficient because at high temperature there are cases when it is not desirable to extinguish a fire: if there is sufficient combustible material in the building the fire should be extinguished before it grows even larger, but if there is little to no combustible material, similar to coals of a fire, there is little gain in extinguishing the fire, as it will cease to burn on its own. Experimental evidence confirms that choosing $Val_k = \frac{\alpha}{D_k \times F_k}$ outperforms both $Val_k = \frac{\alpha \times \pm T_k}{D_k}$ or $Val_k = \frac{\alpha}{D_k}$, where T_k is the temperature of task k.

5.2 Task Evaluation for Ambulances

Ambulances look for buried or wounded civilians. We make the value of civilian k to depend both on its distance from the ambulance agent, D_k, and its level of buriedness B_k:

$$Val_k = \frac{\alpha - B_k}{D_k}.$$

The selected civilian is the one that can be rescued the fastest, because it is both close ot the ambulance and it is not buried deeply. Larger values of B_k mean the civilian is more buried, will lose health faster, and will take longer to be dug out. The scale of distance is much larger than the scale of buriedness. Hence, ambulances will generally choose to uncover the least buried civilian from the closest civilians. This maximizes the number of civilians who are saved if the building catches fire and the ambulance needs to leave. This indirectly coordinates the work of all the ambulances working to dig out the same civilian. When a civilian is unburied, one ambulance will load up the civilian and take it to a refuge while any other ambulance will go to the next most valuable task. As with fire trucks, the value of all the tasks is recomputed at every cycle. Ambulances in general will not switch to a different task while unburying a civilian. This is because new tasks will have a larger distance than the current one, so their value is generally lower.

The approximate health of civilians is visible only in large discrete intervals. This prevents accurate estimations of how long the civilian will survive. Simply prioritizing the rescue of more wounded civilians without knowing how long they will survive causes a high-risk high-reward situation: a civilian either dies before being rescued or is saved due to the prompt arrival of an ambulance. By prioritizing the rescue of civilians who are not buried too much, the ambulance can move more rapidly to the next civilian before their health deteriorates. This rationale is confirmed by experimental testing which showed $Val_k = \frac{\alpha - B_k}{D_k}$ outperforming both $Val_k = \frac{\alpha \pm H_k}{D_k}$ and $Val_k = \frac{\alpha}{D_k}$, where H_k is the percent of total health of civilian k.

5.3 Task Evaluation for Police

Police clear blockages on roads to allow other agents to traverse them. The value of a road segment to a police depends on its distance D_k and how much of it is blocked. Roads that are completely blocked are more important to clear than roads that simply restrict agent throughput. The degree to which each road is blocked, I_k, takes value 1 if the road is completely blocked and decreases rapidly towards zero, never reaching zero, as the size of the smallest choke point increases. Police assign a task k the value:

$$Val_k = \frac{\alpha \times I_k}{D_k}.$$

This makes police agents to select roads that are impassable, even if far away, over roads that are passable but with suboptimal throughput. Once a road is selected as the next task, the police will travel to that road and clear all the blockages on it. As blockages are simply represented as impassable polygons on the roads, there are no additional properties to take into account in their evaluation. Thus, only geometric properties between the blockages and roads can be extracted. Experiments show that the throughput of roads is more important than creating continuous blockage-free roads or simply prioritizing more or less blocked roads, since $Val_k = \frac{\alpha \times I_k}{D_k}$ outperformed $Val_k = \frac{\alpha \times \pm Q_k}{D_k}$ and $Val_k = \frac{\alpha}{D_k}$, where Q_k is the number of blockages on road k.

5.4 Task Selection with no Teams

When there are no teams, agents act independently from each other. Each agent greedily selects the task k with the highest value Val_k using Algorithm 1. We call this method of selecting tasks the “base” configuration, and we use it as a baseline against which we compare the teamwork strategies and other methods. Once a task is selected, the agent will move towards the task and complete it. If there are no known tasks, the agents will explore by moving to a random building or road.

Algorithm 1: Task selection by an agent when there are no teams

Input: $K = \{k_1, k_2, ...\}$; // K is the set of tasks.
if $|K| = 0$ **then**
  return Search for a task;
forall the $k \in K$ **do**
  Compute Val_k ; // as in Section 5.
$k_{max} \leftarrow \text{argmax}_{k \in K}\ Val_k$;
return k_{max};

The independent nature of this strategy makes it simple to implement and robust to lack of communication, but is also its weakness, as there could be many tasks that would benefit from the coordinated effort of multiple agents. Next, we propose strategies that are designed to take advantage of synergies among agents.

6 Working within Teams

Working within a team helps agents prioritize which tasks to select without the need for explicit communication. For example, obstacles will only be cleared in areas where agents are working instead of areas where there are no other agents. Furthermore, inter-type synergies allow agents to be more efficient at doing their tasks. For example, after clearing a blockage, a nearby wounded civilian or a growing fire might be discovered. In this case, by having teams with different types of agents, the travel time for non-police agents to reach the new task is minimized.

When an agent is a member of a team, it evaluates a task based not only on the task's Val_k value (computed in Section 5), but also on the value the task has for the team, as explained next.

6.1 Task Selection within a Team's Area

A team's area is used to implicitly coordinate the agents on a team. All the agents on a team are expected to operate in the team area. The center of the team m, $(\bar{x}_m,\bar{y}_m)$, is computed as the average position of all the agents a on that team:

$$\bar{x}_m = \frac{\sum_{a \in m} XCoord(a)}{|m|}, \bar{y}_m = \frac{\sum_{a \in m} YCoord(a)}{|m|}.$$

where $XCoord(a)$ and $YCoord(a)$ are the x and y coordinates of agent a, and $|m|$ is the number of agents in the team. With a slight abuse of notation, when indicating the distance between a task and a team or between two teams, we use m to refer to the center of team m. Hence, $Distance(k, m)$ is the Euclidean distance between task k and the center of team m, i.e. $(\bar{x}_m,\bar{y}_m)$. Similarly, $Distance(m_1, m_2)$ used in Algorithm 2 is the distance between the centers of teams m_1 and m_2, i.e. $(\bar{x}_{m_1},\bar{y}_{m_1})$ and $(\bar{x}_{m_2},\bar{y}_{m_2})$ respectively.

Teams operate within a radius TR of the team center $(\bar{x}_m,\bar{y}_m)$ by reducing the value of tasks outside this radius. To ensure agents do not leave the team radius, tasks with a value below $Threshold$ are not considered available to an agent. The specific values of $Threshold$ and TR we use are given in Section 9. The task selection algorithm used by an agent in the team is presented in Algorithm 3.

Algorithm 2: Initial team formation using hierarchical clustering

Input: $A = \{a_1, a_2, ...\}, MaxDist, MaxSize$; // A is the set of agents
$Finished \leftarrow \{\{\}\}$; // $Finished$ is a collection of sets
$C \leftarrow \{\{a_1\}, \{a_2\}, ...\}$; // C is the set of clusters, thus a collection of sets
$notDone \leftarrow$ True;
while $notDone$ *and* $|C| \geq 2$ **do**
  $notDone \leftarrow$ False;
  $(C_i, C_j) \leftarrow \text{argmin}_{C_i \in C, C_j \in C}\ Distance(C_i, C_j)$ and $C_i \neq C_j$; // Find the two closest clusters.
  if $Distance(C_i, C_j) < MaxDist$ **then**
    $notDone \leftarrow$ `True`;
    if $|C_i| + |C_j| < MaxSize$ **then**
      $C_{new} \leftarrow C_i \cup C_j$; // C_{new} is the merger of cluster C_i and C_j.
      // Remove individual clusters from C and add the merged cluster.
      $C \leftarrow (C - \{C_i\} - \{C_j\}) \cup \{C_{new}\}$;
    else
      // If the merged cluster was too big, find the larger original cluster.
      $C_{max} \leftarrow \text{argmax}_{C_x \in \{C_i, C_j\}} |C_x|$;
      $Finished \leftarrow Finished \cup \{C_{max}\}$; // Mark this cluster as finished.
      $C \leftarrow C - \{C_{max}\}$; // Remove it from the available clusters C.
$Finished \leftarrow Finished \cup C$;
return Finished

Center agents keep track of the team areas throughout the simulation. Agents send their individual positions to all the agent centers, which then calculate the center of each team area and send it back. If team updates are too infrequent, the center of the team will move very slowly and the team will not explore much of the surrounding area. Frequent updates can cause task thrashing, where an agent moves to a task but the task is outside the team area in the next update. This forces the agent to select a new task and waste more time in transit. To avoid this, team updates are staggered and sent periodically (every 5 time steps in our experiments) to provide low communication overhead and to help avoid task thrashing. To further reduce thrashing, Val_k is modified by a sigmoid function to smoothly devalue tasks near the edge of the team radius TR by using the distance from task k to team center m. The resulting value for the team of task k is computed as follows:

$$TeamVal_{k,m} = Val_k / (1 + e^{Distance(k,m)/TR}) \tag{1}$$

Next, we discuss the hierarchical clustering algorithm our team formation methods use as part of the initial team formation process.

6.2 Hierarchical Team Formation Mechanism

Agents form teams using a hierarchical agglomerative clustering algorithm (Florek et al., 1951). Our main criterion for forming teams is the geographic proximity between agents, unlike other approaches in the literature (Rosenschein and Zlotkin, 1994) that focus instead on the tasks. The reason is that in RoboCup Rescue many tasks are not known until an agent discovers them. Our framework is applicable to any domain where tasks are frequently discovered or appear and that require multiple agents or agent types to complete them. If agents able to work on a newly discovered task are not close to it, then the added travel time may prevent them from completing that task (for example, a wounded civilian might die), or it may slow its completion (for example, if a fire has spread to neighbor buildings). Some tasks require multiple types of agents, for example if a building on fire is surrounded by blockages, the police need to clear the blockages before the fire trucks can get in range to extinguish the fire. By clustering agents according to their spatial position, we partition the map into regions such that if a task appears in a region, the task will be covered by agents in the team assigned to that region. This also makes it easier to track the general locations of agents.

In the team formation process each agent starts as its own cluster in its own initial location. The closest pair of clusters, based on the Euclidean distance between them, attempts to merge. We leave the definition of distance between two clusters general, but for our implementation we define the distance between two clusters to be the distance between the two clusters' centers. If merging creates a resulting cluster of over $MaxSize$ agents, then the larger cluster before the merge is considered a finished team and is left unchanged. This process is repeated until all clusters are finished or until the distance between the closest two clusters is too large for further merging. This ensures that the agents in the team are close to each other while also ensuring that no team is drastically larger than another. Algorithm 2 provides a detailed account of the team formation process.

The initial distribution of agents in the environment does not affect our team formation process, but it affects the performance of the teams once they are formed. When agents start all on one end of the map and tasks on the other, our teams might take longer to explore the entire map than individual agents would do. After teams are formed, they would need to travel across the map to find and complete tasks. The team motion depends on the location of tasks, and can be slower than if individual agents geographically spread themselves at the beginning of the simulation. However, this problem is not related to our team formation algorithms but to the choice we made to favor task completion over geographic coverage of the environment. This trade-off is due to the urgent nature of many tasks. In our experiments we do not change the initial distribution of the agents, which is an integral part of the maps used in RoboCup Rescue, for two reasons: this enables a fair comparison with teams from the RoboCup Competition using the same maps; and the different distribution of agents in different maps allows us to test the robustness of our algorithms in a variety of initial conditions.

7 Team Models

In this section, we describe our three teaming strategies: the case where teams are created at the start of the simulation and do not change over the entire simulation (called "stable"), the case where teams change as required by the tasks (called "fluid"), and the case where the teams have restrictive membership based on domain knowledge (called "split").

Task selection is carried out by each agent in a decentralized manner, but central coordination is required for moving agents to different teams in the fluid and split configurations. The information agents exchange (for example, tasks locations) is susceptible to errors and message delays, so agents do not wait for a central authority to issue task assignments. However, to change team membership, knowledge about other teams is needed. In Section 9.5 we show that this centralized approach is not expensive from a communication point of view.

It is important that the teams have an appropriate size. When teams are too small, the teaming methods do not take full advantage of synergies among tasks. Since the number of agents is limited, if one team is too large then other teams will be smaller and less efficient. To ensure an appropriate size there is an upper limit to the team size, $MaxSize$. The effects of team size are analyzed in Section 8.1.

7.1 Stable Teams

In our stable teaming strategy, teams are formed at the start of the simulation using Algorithm 2 and they do not change. An agent in a stable team chooses its next task using Algorithm 3.

Algorithm 3: Task selection for agent a on a team

```
Input: K = {k_1, k_2, ...}, m ;          // K is the set of tasks, m the team of agent a
forall the k ∈ K do
    Compute TeamVal_{k,m} from Equation 1;
k_max ← argmax_{k∈K} TeamVal_{k,m};
if TeamVal_{k_max,m} < Threshold  or  |K| = 0 then
    return Search for a task;
else
    return Select task k_max;
```

While this approach is simple and takes synergies among agents into account, it lacks the ability to respond to dynamic changes in the environment. In some cases, agents could be allocated elsewhere, where their contribution would be greater. Next, we describe a fluid strategy that also explores synergy among agents, and in addition allows agents to change teams to better respond to dynamics in the environment.

7.2 Fluid Teams

Our approach to forming fluid teams is inspired by ideas outlined in (Rathod and des Jardins, 2004; Gaston and des Jardins, 2005). However, unlike those works which form teams around tasks, our team formation algorithms form teams based on the location of agents.

In a fluid team each agent models the decision problem as a Markov Decision Process (MDP), where each agents gets a reward R for being in a specific state s. The utility for a history of states $(s_0, s_1, s_2, ...)$ is typically computed as a discounted utility, more specifically as the discounted sum of the rewards for each state, i.e, $R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...$, where γ is the discount factor. When $\gamma = 1$, the utility is called additive utility. A purely additive utility is too focused on long-term benefits, while a discounted utility emphasizes short-term benefits since γ decreases exponentially.

To compromise between short-term and long-term benefits, each agent a uses a recursive additive utility function with a bounded range to compute its utility at time t, $U(t)_a \in [U_{min}, U_{max}]$. When the simulation starts, agents are assigned an initial utility halfway in this range, namely $U(0)_a = (U_{min} + U_{max})/2$.

The agent's utility $U(t)_a$ is computed according to Equation 2. This utility depends on the actions taken by the agent. The actions an agent in team m takes depend on the value $TeamVal_{k,m}$ the agent computes for each task k.

$$U(t)_a = \begin{cases} U(t-1)_a + R_d & \text{if the agent is doing a task,} \\ U(t-1)_a + R_m & \text{if the agent is moving to a task,} \\ U(t-1)_a + R_s & \text{if the agent is searching for a task.} \end{cases} \tag{2}$$

Our MDP has three states, depending on what the agent did in the previous time step. If the agent was actively accomplishing a task, it receives the reward R_d. Agents moving to a known task are rewarded R_m. Lastly, when an agent does not know of any tasks or is outside the team's operational area, the agent searches for a task and is rewarded with R_s. The specific values we use for R_d, R_m and R_s are reported later in Section 9. An agent transitions between pairs of these states when it finds a task, when it reaches a task location, and when it finishes completing a task. Agents in a fluid team, as agents in a stable team, choose their individual tasks using Algorithm 3 and update their utility values according to Algorithm 4.

Algorithm 4: Utility update for agent a

Input: $U(t-1)_a, U_{max}, U_{min}$; // $U(t-t)_a$ is the utility at the previous time step, U_{max} and U_{min} are bounds on $U(t)$

Compute $U(t)_a$ from Equation 2 in Section 7.2;
if $U(t)_a > U_{max}$ **then**
    $U(t)_a \leftarrow U_{max}$;
else if $U(t)_a < U_{min}$ **then**
    $U(t)_a \leftarrow U_{min}$;
return $U(t)_a$

Since the RoboCup Rescue environment is very dynamic, an agent's action alone has a small impact on a state transition. Tasks may expire or be completed by other agents, new tasks may appear and the team may move thereby leaving some tasks incomplete. For this reason, the action to transition between states in the MDP is generated from the outcome of the simulation. The agents individual actions outlined in Section 6 are attempts to stay in the most profitable state, but may be forced by the simulation to a suboptimal state.

If a team is limiting the agent's utility by being in a location with few tasks, the agent is able to switch to a different team where there are more tasks. To do this, agents periodically send their current utilities to the agent centers. The centers then calculate three separate average utilities per team, one for each of the agent types. Specifically, for each team m and for each agent type p the centers will calculate:

$$\bar{U}(t)_{m,p} = \frac{1}{\sum_{a \in m | Type(a)=p} 1} \sum_{a \in m | Type(a)=p} U(t)_a \tag{3}$$

If no agent of type p is on the team, the $\bar{U}(t)_{m,p}$ is set to $(U_{max} + U_{min})/2$. These averages are calculated separately since some agent types may have very high utility and others very low utility in certain situations. If there is only one type of task in an area (only fires, blocked roads or buried civilians), agents which can do this task will have a high utility, while agents that cannot do this type of task will have a low utility. The average team utility across all agents depends on the ratio of high vs. low utility agents, but only the low utility agents should be reallocated to a different team.

A short-term expected utility gain method is used to determine whether it will be beneficial for an agent to move to a different team (Algorithm 5). If we define short-term as a time window of W time units, then the expected utility gain of an agent type p going from team m_j to team m_i with an estimated travel time between teams of $TT(m_i, m_j)$ would be:

$$Gain(t)_{m_i,m_j,p} = \bar{U}(t)_{m_i,p} \times (W - TT(m_i, m_j)) - \bar{U}(t)_{m_j,p} \times W \tag{4}$$

The best team m_{max} for an agent on team m_j to move to is defined by:

$$Gain(t)_{m_{max},m_j,p} = \max_{m_i} Gain(t)_{m_i,m_j,p}$$

If $Gain(t)_{m_{max},m_j,p}$ is positive then one agent of type p with the lowest utility $U(t)_a$ on team m_j is transferred to team m_{max}, via a message from the center. Since teams stagger their updates to reduce spikes in communication, this calculation is only done when team m_j (the team possibly sending an agent elsewhere) sends its update.

The time window W needs to be both large enough to allow agents to transfer to teams that are not close, but also small enough to ensure tasks have not expired. Further analysis on the choice of W is given in Section 8.1.

Algorithm 5: Transfer of an agent to a different team

Input: $W, U(t) = \{U(t)_1, U(t)_2, ...\}, P = \{p_1, p_2, ...\}, M = \{m_1, m_2, ...\}$; // W is the time window, $U(t)$ the utility of agents at time t, P the set of agent types, M the set of teams

```
Compute Ū(t)_{m,p} for all m and p using Equation 3 from Section 7.2;
for j = 1 to |M| do
    forall the p ∈ P do
        Compute m_max ← argmax_{m∈M} Gain(t)_{m,m_j,p} using Equation 4 from Section 7.2;
        Compute Gain_max ← Gain(t)_{m_max,m_j,p};
        if Gain_max > 0 then
            // find the agent of type p with the smallest utility on the team
            a_min ← argmin_{a∈m_j} U(t)_a  and  Type(a) = p;
            Transfer agent a_min from team m_j to team m_max
```

One drawback of the way we are forming teams is that the types of tasks that different types of agents can do might be disjoint, and hence having agents of both types on the same team can be detrimental. One agent type might want the team to move in one direction, while the other agent type might want to move in a different direction. This can lead to both agent types to perform sub-optimally. When this happens in fluid teams, these two agent types will end up moving to different teams, so after some time most teams will not have both agent types present. Next, we present a teaming strategy that deals with this issue directly, by restricting the types of agents that can be on the same team from the beginning of the simulation.

7.3 Restrictive Team Membership

Both extinguishing fires and digging out civilians are time consuming tasks that are done by different types of agents, fire trucks and ambulances respectively. Typically, fire and civilian rescue tasks are not both present within a team's radius, so the agent type with no tasks to do will have a low utility and, in the fluid configuration, it will attempt to transfer to a different team. Instead of waiting for one agent type to reach a low utility and transfer, we can create restrictions on team membership in the initial team formation to improve the overall efficiency. We call this the split configuration. We call "restricted agent types" the agent types that are not allowed to be on the same team. For RoboCup Rescue, only fire trucks and ambulance should be restricted. We call them the first and second restricted type respectively. The restrictive initial team formation uses hierarchical clustering in two repeated stages, as described in Algorithm 6, a modified version of Algorithm 2. The algorithm presented is for two restrictive agent types, but can easily be extended to handle any number of restrictive types.

Initially each agent itself is a cluster and all clusters except those containing the second restrictive type are marked as available in the set FP. SP initially contains all clusters except those clusters containing the first restricted type. In the first stage, the two nearest (with respect to distance) clusters in FP are merged as described in Section 6.2.

These two merged clusters are marked as clusters for the first restricted agent type, even if neither cluster contained a first restricted agent type. These clusters are then removed from the pool of clusters available to the second stage, namely SP. The second stage considers all clusters in SP, and again the closest two clusters are merged. The two merged clusters are marked for the second restrictive agent type, and removed from FP if they existed there.

This round-robin hierarchical clustering ensures that teams will be formed based on local proximity while not allowing a team to have both restricted agent types. When an agent center computes the expected utility to transfer an agent to another team it must now check the type of cluster. If the cluster was a first restrictive agent type cluster then $\bar{U}(t)_{m,p} = U_{min}$ for p of the second restrictive agent type and

Algorithm 6: Initial team formation for split teams

Input: $A = \{a_1, a_2, ...\}, MaxDist, MaxSize$; // A is the set of agents

// $Finished$, FP and SP are collections of sets

$Finished \leftarrow \{\{\}\};\ FP \leftarrow \{\{\}\};\ SP \leftarrow \{\{\}\};$

forall the $a \in A$ **do**

  if $Type(a) \neq$ "*second restricted type*" **then**

    // Add agents to FP if they are not of second restricted type

    $FP \leftarrow FP \cup \{\{a\}\};$

  if $Type(a) \neq$ "*first restricted type*" **then**

    // Add agents to SP if they are not of first restricted type

    $SP \leftarrow SP \cup \{\{a\}\};$

$FPnotDone \leftarrow$ `True`; $SPnotDone \leftarrow$ `True`;

while $(FPnotDone\ \ or\ \ SPnotDone)\ \ and\ \ (|FP| \geq 2\ \ or\ \ |SP| \geq 2)$ **do**

  // On the first round try to merge non-second restricted type cluster using Algorithm 7.

  $(FP, SP, Finished, FPnotDone) \leftarrow UpdateCluster(FP, SP, Finished, MaxDist, MaxSize);$

  // On the second round try to merge non-first restricted type cluster using Algorithm 7.

  $(SP, FP, Finished, SPnotDone) \leftarrow UpdateCluster(SP, FP, Finished, MaxDist, MaxSize);$

$Finished \leftarrow Finished \cup FP \cup SP;$

return $Finished$

Algorithm 7: UpdateCluster

Input: $X, Y, Finished, MaxDist, MaxSize$; // X, Y and $Finished$ are collections of sets. $MaxDist$ and $MaxSize$ are constants.

$notDone \leftarrow$ `False`;

// Find the two closest clusters in X

$(X_i, X_j) \leftarrow \text{argmin}_{X_i \in X, X_j \in X}\ Distance(X_i, X_j)$ and $X_i \neq X_j;$

if $Distance(X_i, X_j) < MaxDist$ **then**

  $notDone \leftarrow$ `True`;

  if $|X_i| + |X_j| < MaxSize$ **then**

    // X_{new} is the merger of cluster X_i and X_j

    $X_{new} \leftarrow X_i \cup X_j;$

    // Remove the individual clusters from X and add the merged cluster

    $X \leftarrow (X - \{X_i\} - \{X_j\}) \cup \{X_{new}\};$

    // Remove the individual clusters from Y

    $Y \leftarrow (Y - \{X_i\} - \{X_j\});$

  else

    // If the merged cluster was too big, find the larger original cluster

    $X_{max} \leftarrow \text{argmax}_{X \in \{X_i, X_j\}} |X|;$

    // Mark this cluster as finished

    $Finished \leftarrow Finished \cup \{X_{max}\};$

    // Remove it from the available clusters X

    $X \leftarrow X - \{X_{max}\};$

    // Remove it from the available clusters Y

    $Y \leftarrow Y - \{X_{max}\};$

return $(X, Y, Finished, notDone)$

$\bar{U}(t)_{m,p} = (U_{max} + U_{min})/2$ for p of the first restrictive agent type. A similar method applies for the second restrictive agent type clusters. This ensures both restricted agent types will never be on the same team in the future since it is impossible to transfer if the utility is U_{min}. Agents that are not restricted are still free to transfer to any cluster; they follow the original process.

We have provided algorithms for creating teams that have different properties. In the next section, we present extensive simulation results obtained for our teaming algorithms and the competing methods.

8 Experimental Setup

The simulations were run on the RoboCup Rescue software version from RoboCup 2013 with six different maps: Berlin3, Eindhoven5, Istanbul3, Kobe4, Mexico3, and Paris4. These maps were chosen because of their diversity in terms of geographic layout and connectivity, bandwidth limitations, and because of their reasonably large size. The importance of goals also varies across maps as some maps have difficult fires to extinguish, a large number of civilians to rescue or a large amount of impassable roads. Since the number of available maps is limited, in the competition the same map is used multiple times with different settings, i.e. different number and initial location of agents, different numbers and location of tasks, and different communication channels and bandwidth. These settings change drastically each year to ensure methods are not tailored to specific map settings. The number after the city name indicates the specific settings of the 2013 map used. All the maps and settings are from the final round of the 2013 RoboCup Rescue Agent Simulation League, but centers were added for each agent type if they were not present on a map, namely on Eindhoven5, Kobe4 and Paris4. Each of the configurations was run five times per map and the scores were averaged to account for the randomization in the agents and simulator. Table 1 shows the number of each agent type, buildings, civilians and the number of bytes available for long-range communication for each map.

Map	Ambulance	Fire trucks	Police	Buildings	Civilians	Radio Bandwidth
Berlin3	31	35	26	1426	155	64
Eindhoven5	30	33	30	1308	300	3024
Istanbul3	33	31	30	1244	180	0
Kobe4	30	30	30	736	300	2560
Mexico3	50	46	50	1556	205	3024
Paris4	30	36	31	1618	300	200

Table 1: The number of agents of each type, buildings, civilians and communication bandwidth (in bytes) for each map tested.

The official score for the RoboCup Rescue Agent Simulation League is computed as

$$(CA + CH) \times \sqrt{(BD)}, \tag{5}$$

where CA is the number of civilians alive, CH is the average percent of health left for all civilians and BD is the average percent of building health. Since the civilians can only lose health and buildings can only burn down, the score monotonically decreases as the simulation progresses. The final score is a product between building and civilian health, which means that scores are penalized for focusing on just one of these aspects. Since police do not interact with fires and civilians, they do not directly affect the score. However, clearing road blockages is critical since many agents start being trapped and fires may grow too large while rescuers look for a path around blockages. The maximum score on each map is $CA + 1$. The score function is the standard metric for comparing results in RoboCup Rescue Agent Simulation.

Our results report the percentage of civilians and buildings saved, which are used to measure the task completion rate of ambulances and fire trucks, respectively. We do not use the number of buildings extinguished because more buildings can be extinguished by purposely letting a fire grow, which is not a desired goal. The efficiency of police agents is hard to evaluate since they play a supportive role. For instance, counting the number of blockages cleared is not a good metric since not all blockages need to be cleared. If there are no civilians or fires in a section of the city, clearing blockages there provides no benefit.

Next, we describe the tuning of the parameters for our team configurations. This is followed by a detailed analysis of the results on three of the six maps, an overall analysis of the results and then by an analysis of the communication costs.

8.1 Parameter Setting

There are half a dozen parameters for the teaming configurations that need to be optimized for different domains. We give a general description of the impact of changing these parameters to provide an intuition on how they affect the team configurations. Specifically, we analyzed the maximum number of agents initially on a team, $MaxSize$, maximum distance agents can move to join a team in initial formation, $MaxDist$, the team's radius for selecting tasks, TR, and the time window for the utility gain when changing team membership, W, across all six maps with five repetitions for five distinct values for the split configuration.

There is an interaction effect between the maximum team size, $MaxSize$, and the maximum distance to initially join a team, $MaxDist$, since both indicate when the hierarchical clustering should stop merging teams for initial formation. Ideally, teams should be of a moderate size, since if the team is too small, agents will have difficulty completing tasks that require multiple agents. Large teams are also problematic, since this puts a large proportion of the agents in the same geographic location and thus greatly reduces the overall exploration of the map. Tasks within the team radii will be completed quickly leading to low utilities, while tasks outside team radii will be unavailable to agents.

	$MaxSize = 3$		$MaxSize = 5$		$MaxSize = 7$		$MaxSize = 9$		$MaxSize = 11$	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Berlin3	22.02	1.74	39.07	1.49	**40.42**	**1.74**	39.04	2.57	27.98	2.81
Eindhoven5	36.09	7.50	48.16	5.92	49.48	4.00	**52.06**	**4.49**	37.11	9.60
Istanbul3	16.96	1.97	18.87	3.54	**26.18**	**1.15**	22.23	6.51	16.11	1.83
Kobe4	55.00	3.16	59.90	1.62	**60.41**	**1.56**	58.41	1.97	54.46	6.27
Mexico3	63.48	1.05	69.98	0.66	73.60	0.48	**74.75**	**0.80**	73.21	0.85
Paris4	19.44	3.17	22.79	3.95	**30.76**	**1.86**	22.90	2.97	13.54	3.70

Table 2: Percent of maximum score using the split configuration when changing the initial maximum number of agents per team, $MaxSize$.

	$W = 7$		$W = 10$		$W = 13$		$W = 16$		$W = 19$	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Berlin3	28.25	9.05	40.42	1.74	41.63	1.97	**42.30**	**2.33**	39.39	5.06
Eindhoven5	31.05	5.58	49.48	4.00	**50.40**	**3.62**	49.30	3.38	46.99	4.97
Istanbul3	22.34	3.49	26.18	1.15	27.39	1.25	**29.73**	**1.38**	23.55	5.44
Kobe4	57.01	2.59	**60.41**	**1.56**	59.89	1.88	60.11	1.70	52.73	4.11
Mexico3	71.02	1.06	73.60	0.48	**74.00**	**0.52**	73.07	0.75	71.29	2.50
Paris4	21.89	3.35	30.76	1.86	30.43	3.84	**31.68**	**1.05**	26.92	3.98

Table 3: Percent of maximum score using the split configuration when changing the utility gain time window, W.

The effect of $MaxDist$ is very dependent on the value of $MaxSize$, since even if $MaxDist$ is arbitrarily large teams will be limited to a fixed maximum number of agents, and the clustering process will generate the same clusters once $MaxDist$ is large enough. Given this dependence, we varied $MaxSize$ from the default value of 7, to better control the termination of the hierarchical clustering. We can see from Table 2 that for very small and large sized teams, the split configuration performs significantly worse than for medium sized teams. The exception of this trend can be seen on Mexico3 since it is a medium sized map and has approximately 70% more agents available than the other maps. We also see that on larger maps, such as Berlin3 and Eindhoven5, creating teams of $MaxSize = 11$ significantly decreases the score, while on smaller maps, such as Kobe4, the score is not as drastically impacted by the larger teams. This shows that both the

	$TR = 50$		$TR = 200$		$TR = 400$		$TR = 600$		$TR = 1000$	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Berlin3	24.41	4.67	29.06	10.37	**40.42**	**1.74**	38.54	3.31	36.11	5.04
Eindhoven5	24.04	3.50	46.03	6.06	**49.48**	**4.07**	45.17	3.51	41.08	2.79
Istanbul3	18.37	3.02	23.14	1.56	**26.18**	**1.24**	25.47	1.10	16.17	3.53
Kobe4	28.68	4.12	56.76	4.17	**60.41**	**1.56**	56.15	4.40	46.67	4.96
Mexico3	33.27	11.47	62.83	6.48	73.60	0.48	**73.84**	**1.05**	73.58	0.96
Paris4	14.74	2.37	27.84	1.23	**30.76**	**1.86**	26.61	3.55	15.56	4.87

Table 4: Percent of maximum score using the split configuration when changing the team radius, TR.

	$TR = 50$		$TR = 200$		$TR = 400$		$TR = 600$		$TR = 1000$	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
W=7	22.81	5.73	26.06	6.16	**28.25**	**9.05**	27.67	4.57	25.48	5.83
W=10	24.41	4.67	29.06	10.37	**40.42**	**1.74**	38.54	3.31	36.11	5.04
W=13	24.36	4.81	32.96	4.23	**41.63**	**1.97**	41.24	2.37	33.22	4.58
W=16	26.29	5.95	35.28	4.91	42.30	2.33	**42.71**	**3.52**	38.62	6.28
W=19	35.47	6.28	33.93	6.96	**39.39**	**5.06**	37.64	5.35	36.99	5.25

Table 5: Percent of maximum score on Berlin3 when changing both the team radius, TR, and gain window, W.

map size and the number of agents available should be considered when setting $MaxSize$ and $MaxDist$.

As mentioned above, it is important to have enough teams so that their radii (TR) covers most of the map, as seen in Table 4. However, a larger team radius reduces the coordination between agents and, in the extreme case where a team radius covers the whole map, the team configuration is essentially reduced to the base configuration. A smaller team radius emphasizes the coordination between agents. However, if the radius is too small then the agents will not have enough tasks available, and will instead just focus on exploring the environment. Thus, it is important to have enough teams so their radii cover most of the map while also having the radii small enough that coordination can take place over a reasonable number of tasks.

The utility gain time window (W) determines how easy it is for an agent to transfer to a team with a higher utility. From Table 3 we can see that varying this parameter exhibits a similar pattern to $MaxSize$, as extreme values of W decrease the score. If W is prohibitively small, agents will be very unlikely to transfer to other teams, and if $W = 0$ then whatever configuration is being executed would reduce to the stable configuration. Indeed, we can see that when $W = 7$ the scores get closer to the stable configuration, despite the membership restriction on the split configuration. However, when W increases the score is not reduced substantially, yet the standard deviation typically increases. This indicates that the transfer of agents between teams can be beneficial if tasks still exist when an agent arrives, but can also be detrimental as tasks might be completed before the agent arrives (or when the agent spends too much time traveling instead of exploring locally).

An evaluation of changes in both the team radius, TR, and the utility gain time window, W, for the Berlin3 map is presented in Table 5. To investigate whether there is an interaction effect between TR and W, we fit a model using polynomial regression on both variables. A nested model F-test confirms that this polynomial regression is a better fit than a linear regression and yields a p-value of 1.571×10^{-6}, which also supports our observation that extreme high or low values of these variables reduce the score. In the polynomial regression all interaction effects are statistically significant, which indicates that these parameters should be optimized together instead of individually. There is still a large variation in optimal parameter values between maps, which depends on visible map properties, such as size, road connectivity and agent availability, as well as unknown map properties, such as task distribution.

The default settings in our simulation used $MaxDist = 1,000$ meters, which is about five blocks. For the maximum team size, we found that $MaxSize = 7$ worked well to prevent inefficiencies in completing tasks. A radius of about two blocks ($TR = 400$ meters) was found to be a good operational range to implicitly promote cooperation while providing enough available tasks for each team. When tasks fell below $Threshold = 10^{-5}$, they were not considered available to agents. All not extreme values of $Threshold$ tested yielded no noticeable difference in scores. If $Threshold = 0$, agents will leave the team radius and act similar to the base configuration. When $Threshold$ is too high, agents will disregard available tasks within the team radius. Good values of coefficients for utility rewards were empirically found to be: $R_d = 4$, $R_m = -1$ and $R_s = -2$ with bounds $U_{min} = 0$ and $U_{max} = 100$. The time window, W, for computing the short-term expected utility gain is $W = 10$.

It is very difficult to optimize any of these parameters with the information available at the beginning of the map, so we empirically determined values that worked well across multiple maps. $MaxDist$ and $MaxSize$ create the initial teams, so with the known initial location of agents it would be possible to plan initial teams using map information. However, very few of the tasks are known initially so it is difficult to know how teams should be formed for the unknown tasks. The team radius, TR, is dependent on the known map size, but also on the distribution of tasks which is not known initially. The time window, W, should be optimized based on the distance between teams, but since teams will move to tasks this also cannot be very well optimized based on map information. The utility rewards and bounds are more dependent on TR and W than geographical features of the map.

We used these parameter values to run experiments for each of the proposed teaming configurations. We present the results of these experiments next.

9 Simulation Results for our Configurations

In the following analysis, we compare five configurations:

1. "sample" configuration, which is provided with the simulator;
2. "base" configuration as described in Section 5;
3. "stable" configuration as described in Section 7.1;
4. "fluid" configuration as described in Section 7.2;
5. "split" configuration as described in Section 7.3.

Our configurations do not assume any knowledge of when and where tasks will appear or disappear, since that knowledge is not available in RoboCup Rescue. All the analysis in this section is post simulation analysis.

9.1 Berlin Map Results

Berlin is a large city with many small fires distributed throughout the map. Since smaller fires are easier to extinguish than larger fires, it is important to quickly find all the fires and extinguish them simultaneously. This makes exploring very important initially, and since few tasks are known at the start most agents do explore. The benefits of exploring are reduced as many tasks exist initially but are unknown, while fewer tasks actually appear during the simulation.

The base configuration finds these fires faster than the teaming strategies, but sends too many fire trucks to the first few fires seen. This causes the fire trucks to traverse the large map multiple times in order to extinguish all the fires and causes inefficiency. A fire in the top left of the map goes unnoticed for a while in all teaming configurations, causing damage to a fair portion of the city in all configurations as seen in Figure 1 and Figure 2. The fire in the bottom center is noticed early, but the roads nearby have blockages so good coordination is needed between police and fire trucks.

This map also has low communication bandwidth, which affects the teaming configurations less than the base configuration. The base agents are not able to send all the information due to the communication restrictions, so many agents miss crucial information such as the locations of civilians or fires. The teaming configurations use the limited available communication bandwidth for organizing teams and simply share map information locally. Since teams operate in a restricted spatial area, many team members have similar knowledge of the environment so implicit coordination occurs. The base agents also share information locally, but agents of different types are much more spread out making synergies more rare.

The stable teams benefit from some synergy between agents, but often agents do not have any available task inside the team's area. This lack of full utilization of agents significantly hurts the score compared to the fluid teams, which transfer agents to different teams to ensure there are available tasks. Eventually the team compositions of the fluid teams and split teams become very similar, but in the fluid team configuration the fires have expanded much more while agents were trying to find the best team compositions.

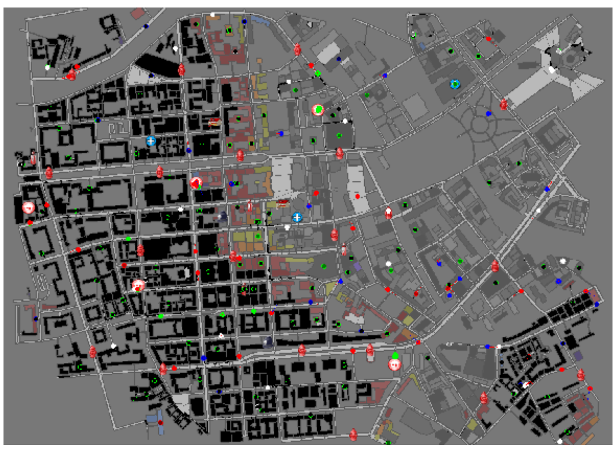

Figure 1: Berlin at the simulation end for sample configuration.

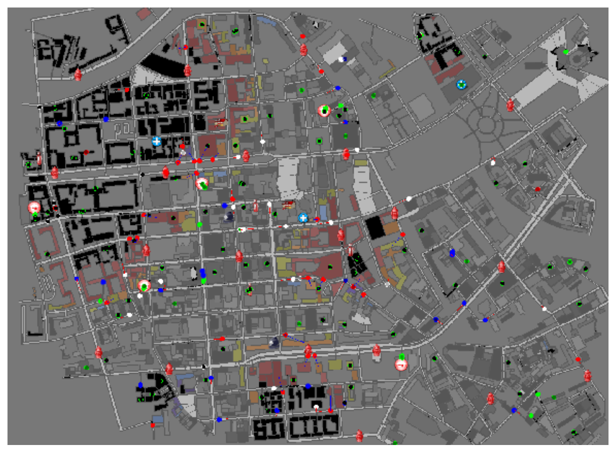

Figure 2: Berlin at the simulation end for split configuration.

9.2 Kobe Map Results

Kobe is a much smaller map than Berlin, both in terms of number of buildings and distance to travel across the map. There is a large fire in the bottom left portion of the map that is important. This fire is noticed quickly since agents start nearby, but the area around the fire has many blockages which make it difficult for other fire trucks to help out. In the Kobe map settings the fire trucks extinguish significantly slower compared to Berlin.

Figure 3 and Figure 4 show the city Kobe at the end of the simulation for the sample and split configurations respectively. From the figures it is clear that the split configuration saves many more buildings from burning down than the sample configuration, but less evident is that the split approach also rescues more civilians. Fires not only destroy buildings, but also rapidly hurt civilians who are trapped inside a burning building. Fires also self-replicate themselves in a positive feedback loop; more fires cause nearby buildings to catch fire faster. This causes fires to have a large effect on the score, thus it is crucial to contain fires. This can often cause a bimodal distribution in scores based on whether the fires on the map are contained or not. This bimodal effect is discussed at greater detail in Section 10.

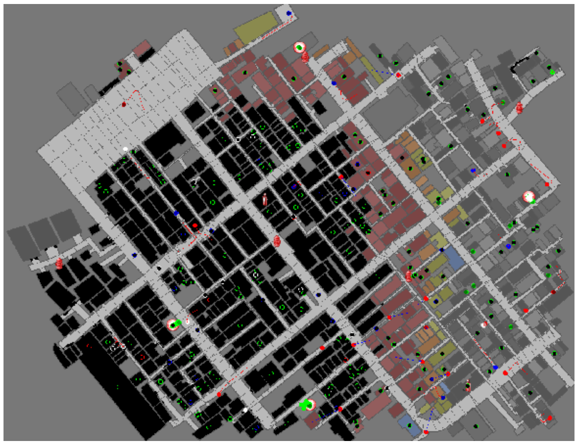

Figure 3: Kobe at the simulation end for sample configuration.

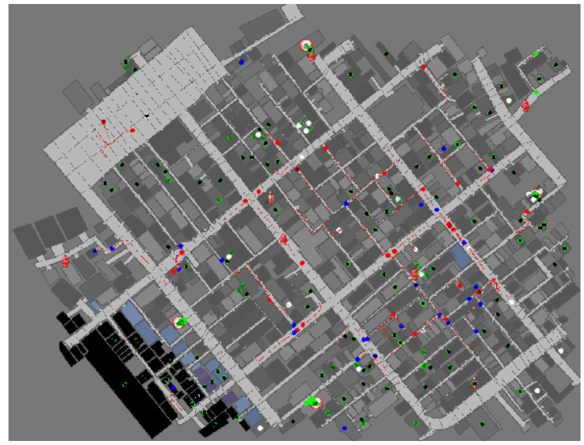

Figure 4: Kobe at the simulation end for split configuration.

There is a large amount of long-range bandwidth available on this map, so communication is not very restrictive. The base configuration agents know about the important fires, but without a coordinated effort from both fire trucks and police containing the fire is difficult. This causes the larger standard deviation in the score for the base configuration shown in Table 8 in Section 9.4, since some times the fire grows uncontrollably similar to Figure 3 and other times it is contained as in Figure 4. Since the Kobe map is small compared to most other maps, the team's task selection radius covers a larger portion of the map. This lets multiple teams reach the crucial fire, even when the stable team configuration is used. The fluid teams transfer agents to other teams, instead of overlapping them as in the stable team configuration. Transferring agents is based on their utility is more flexible and scores slightly higher than the stable configuration.

9.3 Mexico City Map Results

Mexico City is a medium size map that has a large number of available agents and a large communication bandwidth. This makes the map fairly easy and all configurations, except sample are able to retain a large portion of the original score. In the sample configuration much of the city burns down, creating a similar situation to the Kobe map. There is a medium size fire in the bottom center of the map initially, but some police are nearby and can quickly report the fire to the fire trucks using the abundantly available communication bandwidth. Since the sample configuration agents do not communicate, many fire trucks explore the map instead of helping extinguish the initial fires, which ends up causing significant building damage.

By communicating the positions of the initial fires, the base and other team configurations quickly extinguish all initial fires on a map along with any new fires that spring up along the way. The base configuration is the best at extinguishing fires, since half a dozen police agents start next to the initial fires. Thus, base configuration police agents are able to rapidly remove blockages near the fires, since they simply clear the closest known blockage. The implicit coordination in all our teaming configurations also helps in clearing these blockages, but only after the police rendezvous with their teammates. Since the fires are relatively easy on this map and there is an abundance of agents, the fires are extinguished rapidly even with poorer team compositions and before the utilities have stabilized across teams. This causes all the teaming configurations to perform similarly in terms of fire damage. For all the configurations except the sample agents, the end of the simulation looks similar to the Kobe split configuration in terms of building damage. Therefore, the main challenge in this map is to maximize the amount of civilians rescued.

In the base configuration, the lack of coordination between police and ambulances makes reaching civilians and exploring efficiently much more difficult. All our teaming configurations are able to save approximately the same number of civilians, since most of the time there are no active fires in the map. This means that fire trucks will have low utility on all the teams and, even on stable teams, ambulances will be able to direct the team since they have active goals while the fire trucks are simply exploring.

9.4 Analysis of Experimental Results

To provide a baseline for comparison, we tested our teaming configurations against both the sample agent which is provided with the RoboCup Rescue Agent Simulator and two state-of-the-art algorithms from the literature. The algorithms from literature are max-sum and an auction method (Nanjanath et al., 2010). The auction method is a sequential single-item auction, which plans task assignments over the long term. This allows comparison to our greedy task selection methods. Max-sum can be efficiently implemented and has been shown to yield good solutions to various sub-problems in RoboCup (Ramchurn et al., 2010a; Pujol-Gonzalez et al., 2014). This also allows a comparison between a fully decentralized solution, max-sum, our partially decentralized solution and the fully centralized auction method.

Both of the methods from the literature are very dependent on communication to assign tasks, and typically perform poorly when the communication is limited. The Istanbul3 map has no communication channel, which would make both of these algorithms unable to assign tasks to agents. Instead these two algorithms benefited by using the communication settings for Berlin3 on this map, which provides a very low bandwidth channel.

Table 6 and Table 7 show the percent of civilians rescued and percent of buildings unburnt respectively. The RoboCup scoring, as described in Section 8, is shown in Table 8. This scoring function encourages algorithms to save both civilians and buildings and penalizes when either aspect is insufficiently handled. The best configuration for each map is shown in bold. The fluid configuration does slightly worse than split, but better than the others. This shows that forming teams based on geographic area to induce implicit cooperation is helpful.

	Sample		Auctions		Max-Sum		Base		Stable		Fluid		Split	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Berlin3	39.48	2.25	32.44	5.20	33.08	4.85	51.23	2.17	50.32	1.02	**51.87**	**1.34**	51.23	1.08
Eindhoven5	**73.00**	**2.27**	57.87	10.35	66.71	5.10	68.47	2.04	64.13	0.77	72.80	0.56	72.73	1.53
Istanbul3	42.11	1.77	50.17	5.29	50.28	7.12	46.44	2.24	**56.00**	**1.86**	44.44	1.11	46.78	1.38
Kobe4	50.20	1.61	58.27	3.43	**63.19**	**2.60**	60.67	3.37	61.40	1.26	61.27	0.43	62.67	0.71
Mexico3	56.68	1.63	59.71	5.70	69.03	2.26	73.95	1.78	77.17	1.48	76.39	1.94	**77.95**	**1.52**
Paris4	19.40	0.60	21.13	6.95	22.19	3.83	26.27	0.72	30.93	1.53	33.87	0.77	**34.27**	**0.72**

Table 6: Percent of civilians alive at end of simulation for different teaming configurations on various maps.

	Sample		Auctions		Max-Sum		Base		Stable		Fluid		Split	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Berlin3	34.45	4.43	31.37	8.10	36.42	9.15	37.17	5.48	31.76	4.38	45.38	4.86	**60.45**	**3.70**
Eindhoven5	12.34	5.38	14.64	7.77	14.23	2.97	38.33	5.90	6.34	2.39	39.17	4.04	**47.63**	**4.17**
Istanbul3	16.25	4.21	25.19	1.53	21.77	3.23	16.47	4.71	16.06	4.83	27.55	5.57	**29.94**	**5.82**
Kobe4	30.57	8.49	61.19	8.58	62.28	5.61	57.48	11.03	80.59	3.68	**89.83**	**3.41**	89.12	2.98
Mexico3	28.72	4.48	39.41	5.02	45.30	3.35	**98.34**	**1.04**	87.01	3.31	86.70	3.06	86.83	2.73
Paris4	7.30	3.54	35.66	20.19	32.54	12.42	21.33	4.36	46.61	5.75	60.87	5.66	**81.58**	**3.88**

Table 7: Percentage of buildings unburnt at end of simulation for different teaming configurations on various maps.

The sample configuration is by far the worst of the configurations tested. Agents only rely on local information and do not communicate. When they have no more tasks in their local information, agents start

	Sample		Auctions		Max-Sum		Base		Stable		Fluid		Split	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Berlin3	22.77	1.68	18.16	4.66	19.73	3.25	31.36	0.97	28.99	1.58	35.52	1.92	**40.42**	**1.74**
Eindhoven5	25.70	1.44	22.14	9.29	25.20	4.61	42.22	1.94	16.32	3.76	45.92	2.69	**49.48**	**4.07**
Istanbul3	17.00	2.19	25.35	2.72	23.59	4.67	18.72	1.36	22.69	2.71	22.87	1.22	**26.18**	**1.24**
Kobe4	27.45	1.64	45.35	5.63	49.95	3.60	46.51	6.30	56.14	1.22	57.92	1.16	**60.41**	**1.56**
Mexico3	30.63	1.03	37.41	5.78	46.22	2.72	73.33	0.49	70.52	0.66	72.22	0.34	**73.60**	**0.49**
Paris4	5.19	2.21	13.14	7.83	12.74	4.80	12.28	5.54	21.37	5.46	26.83	2.62	**30.76**	**1.86**

Table 8: Percentage of maximum possible score for different teaming configurations on various maps compared against the Sample agents provided with the simulator and two algorithms from the literature.

walking around randomly. This allows agents to explore the map quickly, but does not use agents efficiently since the vast majority of their time is spent exploring. All other configurations share newly discovered tasks with other agents.

The auction method plans tasks for agents to optimize over time, but also requires more communication. On Berlin3, Istanbul3 and Paris4 the communication bandwidth is too limited and some messages cannot be sent, which delays the assignments. Initially, much of the environment is unknown and new tasks are frequently discovered. This causes the assignment plans to change drastically at beginning of the simulation. When only few tasks are known, agents will cluster around these tasks and move away from the other parts of the map. Agents then take longer on average to reach newly discovered tasks as agents are not spread out on the map. Later in the simulation the planning changes less frequently, but by this point the method has fallen significantly behind the better configurations.

Max-sum also tends cluster agents via assignments when few are known, but not as much as the auction method. While max-sum has been shown to produce good results, this has been done by considering only one agent type at a time. This formulation is unable to consider synergies between agent types. This is seen as police clear blockages at a faster rate than the teaming configurations, yet the performance of the ambulances and fire trucks is lower, as seen in Table 6 and Table 7. This indicates that the police are clearing blockages in non-crucial areas.

The base configuration also does not coordinate the police with ambulance and fire truck agents. The greedy task selection used spreads out the agents more evenly across the map initially than either max-sum or the auction approach. This enables the agents to react more rapidly when new tasks are found, but also is more inefficient when most of the tasks are known. This causes the base configuration to perform comparably to both max-sum and the auctions method.

Stable teams do not do well because agents are not used efficiently. One strength of stable teams is attempting to rescue civilians near fires since the fire trucks can keep the burning buildings away from ambulances long enough to rescue civilians. The problem with this approach is that many fire trucks are on teams with ambulance agents rescuing civilians not near fires. This inefficiency in the use of fire trucks causes a vast portion of the city to burn down where teams are not present and burns many civilians before agents become aware of their existence.

Fluid teams overcome this inefficiency by allowing agents to move to a team where they will be utilized more. Teams tend to accumulate either fire truck or ambulance agents and lose the other type. Police are the agent type that changes team the most frequently, due to their faster task completion rate. Fire trucks can spend the entire simulation trying to contain one fire, but police will run out of roads to clear fairly quickly, depending on the team radius size. These unused police agents will switch to teams that are still mobile and encountering new impassable roads. The ability to switch teams substantially increases the average utility of agents and as a result the fluid configuration always performs better than the stable configuration.

The overall best configuration is the split configuration, which was inspired by observing the disparity in the efficiency of agents on the same team. This configuration still retains the fluid team membership aspect allowing all agents to move when this increases the utility.

9.5 Analysis of Communication Costs

In the RoboCup Rescue simulator, there are two major forms of communication. One method, which mimics communication via shouting or megaphone, has a limited range but a large bandwidth for the size of the messages sent. The second method, which corresponds to radio communication between agents, is through channels. Any message sent on a channel is heard by everyone who is tuned to that channel, but the bandwidth on the channels is limited, as seen in Table 1. Agent centers are the only ones who can listen to multiple channels at the same time, so they can act as communication hubs. However, any message sent on any channel can get dropped or corrupted at any time to represent communication failure.

	Ambulance		Fire truck		Police		Team	
	μ	σ	μ	σ	μ	σ	μ	σ
Sample	0	0	0	0	0	0	0	0
Base	43.82	28.92	23.33	30.81	3.70	5.01	0	0
Stable	43.14	31.07	27.43	38.06	3.39	1.89	2.54	1.89
Fluid	45.88	30.39	19.23	26.88	3.17	4.64	2.64	1.93
Split	45.46	30.43	25.60	33.28	3.67	4.64	2.28	1.98

Table 9: The number of messages communicated at each time step averaged across all the maps and experiments.

If there were no failures, the average number of messages the agents attempt to share at each time step in the simulation is shown in Table 9. Ambulance messages share the location of trapped civilians, fire truck messages inform others if a building catches fire or is extinguished, police messages list critical road blockages that need to be cleared, and team messages are the sharing of utility and position to coordinate the teams and allow transfers. The team messages are only approximately 1% of the overall messages, yet the teaming configurations yield large benefits. The standard deviations are large for most message types because the number of messages is averaged across all maps and map settings can have a large effect on the types of messages generated. On difficult maps such Berlin3 in Section 9.1 where fire engulfs the city, there is a large amount of fire truck messages and few ambulance messages since many civilians perish. However on maps where fires are easily extinguished, primarily ambulance messages are generated.

The average message size is 10 bytes, so it is very easy to flood radio channels on maps such as Berlin3 and Paris4. The team configurations perform especially well in these cases, since the environment information messages can be communicated via the short-range method, using the long-range radio channel exclusively for team communication.

10 Evaluation of Robocup Competition Results

The biggest effect on the score obtained by agents in RoboCup Rescue is the inherent difficulty of the map settings. Maps typically fall into one of three categories: difficult, challenging, and easy. Berlin3 (Section 9.1), Kobe4 (Section 9.2), and Mexico3 (Section 9.3) are examples of difficult, challenging, and easy respectively. As noted in (Parker et al., 2014), it is difficult to compare precisely different algorithms, since on easy maps most algorithms will score high regardless and on difficult maps the city will typically burn down and all algorithms tend to perform poorly. Even when comparing algorithms on the competition maps, which are built to avoid maps that lack discrimination power, it can require thousands of simulations to reach statistical significance between different agent configurations. This is due to the large variance in the scores. If the maps settings are too hard or easy, substantially more simulations are needed before statistical significance can be reached. Much of this variance is due to the limited visibility of the agents, and the fact that the various strategies for exploring the maps affect the performance based on how the tasks are placed in comparison to the exploration assumptions.

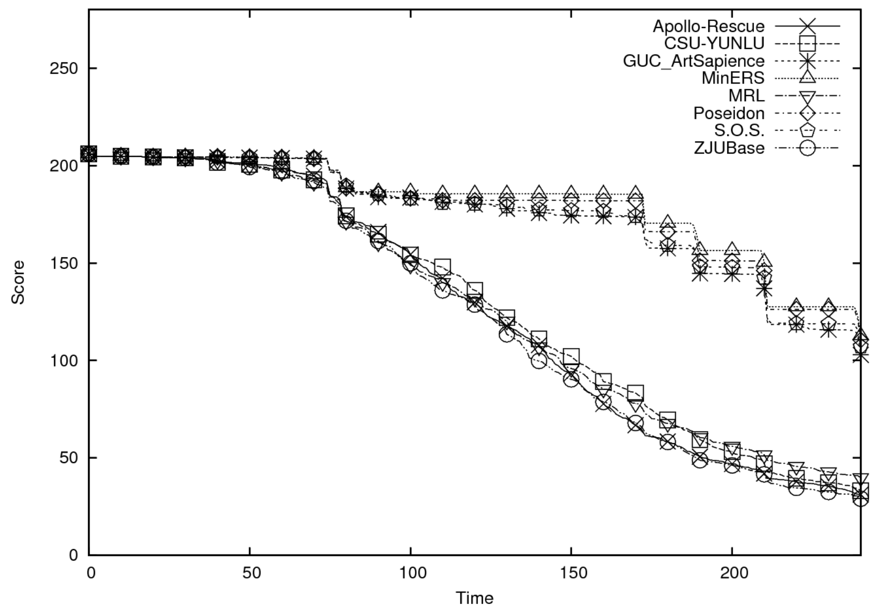

Figure 5: Bimodal score time series of 2013 competitors on Eindhoven3 map.[1]

On challenging maps, such as Eindhoven3 shown in Figure 5, there can be a large difference in score while the difference in algorithms is not significant. This is because an algorithm will score much better if it is able to barely contain a fire compared to when the fire slowly engulfs the whole city.

On the Kobe4 map, the five runs of the base configuration produced scores of 41%, 44%, 44%, 47% and 57% of the maximum. The 57% score is on par with the stable and fluid team configurations since in this case the base configuration was able to contain the fire. Even though this is the same configuration, randomness can have a substantial impact on score for bimodal maps. Despite these issues with the non-normal nature of scoring in RoboCup Rescue, applying the Wilcoxon signed-rank test shows significant difference in the scores of the split configuration compared the other configurations with a p-value of 0.03164. The Wilcoxon signed-rank also indicates that the sample configuration is significantly worse than both the base and fluid configurations. The stable configuration is significantly worse than fluid. The fluid configuration did not quite outperform the base configuration with a p-value of 0.05246. From this we can conclude that split configuration performs better on average than the other configurations and algorithms. We can also conclude that the ability to transfer agents to different teams significantly increases the score.

Although the percent of maximum scores from Table 8 indicates the performance on a map, it is difficult to determine how much of the performance is due to the algorithms compared to the map settings. For example, Istanbul3 from Table 8 has a low score, but our split team performs better than three of the four finalists on this map. This would indicate Istanbul3 is a hard map and not a problem with our teaming configurations. To rank the competitors, in the RoboCup competition the score obtained by each team is mapped to a number of points, which depends on the scores of the other competitors. These points are then summed across all maps and the team with the highest number of points in each round is the winner. As described in (Parker et al., 2014), assigning points for each team is difficult due to the different maximum scores across maps and the non-normal distribution of scores within a map. This makes the ranking method used in the competition not adequate to assess the real effectiveness of the algorithms used by the different competitors and to correctly rank them. The usage of non-parametric statistical tests in this paper avoids the shortcomings of the point system used in RoboCup. More details are given in (Parker et al., 2014).

Using the ranking system from RoboCup 2013, the split configuration would tie for third place against the 2013 finalists, with all the final round maps. As the methods the finalists use are not documented, we are unable to run a more rigorous comparison outside of the logs from the competition. While there are not enough maps to reach a statistical significance in determining the ranking, this demonstrates that with a little domain knowledge the teaming algorithms perform comparably to the RoboCup 2013 finalists. Overall, our results indicate teamwork increases the efficiency of agents, and even without much specific tweaking to the simulator, the split team configuration made it to the semifinal round in the 2013 RoboCup Rescue Agent Simulation League competition.

11 Conclusions and Future Work

We proposed domain-independent algorithms for team formation, which use hierarchical clustering based on the physical location of the agents to create different types of teams, i.e., stable teams that are fixed for the whole simulation, fluid teams that allow agents to transfer between teams, and split teams, which limit the type of agents in each team. The different types of teams are evaluated in the RoboCup Rescue simulator and compared to each other, to a baseline, and to two state of art coordination algorithms.

Our results show that our teaming algorithms are more effective than the baseline and the two competing algorithms. Specifically, our split configuration outperforms all the other algorithms we tried. Our results provide evidence of the tradeoff between myopic methods, such as the ones used by the base agents, that allow for quick decision-making with lower quality allocation and less myopic methods, such as the teaming methods, that produce better allocations but might operate on inaccurate information because they take more time to make decisions.

We also found that forming teams help agents complete tasks more efficiently when agents can move more freely. When comparing the results of our base and static configurations we see that forming teams is not as advantageous if teams over-restrict agents' area of operation. The static configuration does not allow agent transfers, hence agents miss on opportunities to move to areas where they could complete more tasks. However, if transfers are allowed, the constraints on where agents can operate are more relaxed; this is when forming teams has a clear advantage (fluid and split configurations vs. base configuration).

The main reason why the split configuration has an edge over the other algorithms is that it takes agent synergies into account when forming teams. The split configuration is based on the observation that fire and ambulance agents have tasks that are unique to them and that require more time to be completed, hence these agents have the potential to push teams into different geographical areas. This finding suggests that when forming teams it is imperative to find out which agent types are complementary, and which are dominant, and how the interaction among dominant agent types affect teamwork.

Future work will investigate mixed-initiative architectures in which human and agents collaborate. We envision humans working as center agents, and using the information gathered to make the decisions that in RobopCup Rescue are currently made by the agent centers. The research questions here would focus on how humans make decisions and how they would share decision making and responsibilities with autonomous agents (van Santen et al., 2009). We want to explore how different levels of autonomy affect the effectiveness of the humans and the agents, and how much burden these autonomy levels place on the human agents (Harriott et al., 2011). In addition, we would use inspiration from humans to improve decision making in autonomous agents. Better autonomous agents could then be integrated in the simulators used to train first responders and public safety personnel, where they will enrich the training experience and make it more

[1]From http://roborescue.sourceforge.net/

effective (Kincaid et al., 2003).

Further future work will compare the performance of our fluid and split teams to the performance of ad-hoc teams (Stone et al., 2010). While ad-hoc teams have been explored in other domains, to the best of our knowledge they have not yet been extended to RoboCup Rescue, where they could be advantageous due to the dynamic way in which teams are formed. Another interesting question is how synergies among agent types would affect the formation of ad-hoc teams.

References

Akin, H. L., Ito, N., Jacoff, A., Kleiner, A., Pellenz, J., and Visser, A. (2013). Robocup rescue robot and simulation leagues. *AI Magazine*, 34(1):78–86.

An, B. and Lesser, V. (2010). Characterizing Contract-based Multi-agent Resource Allocation in Networks. *IEEE Trans. on Systems, Man, and Cybernetics: Part B*, 40(3):575–586.

Bertsimas, D. J. and Weismantel, R. (2005). *Optimization over integers*. Dynamic Ideas, Belmont, MA.

Chevaleyre, Y., Dunne, P. E., Endriss, U., me Lang, J., Lemaître, M., Maudet, N., Padget, J., Phelps, S., Rodríguez-Aguilar, J. A., and Sousa, P. (2006). Issues in multiagent resource allocation. *Informatica*, 30:3–31.

Choi, H.-L., Brunet, L., and How, J. (2009). Consensus-based decentralized auctions for robust task allocation. *IEEE Trans. on Robotics*, 25(4):912 –926.

Cooper, D. C. (2005). *Fundamentals of Search and Rescue*. Jones and Bartlett Publishers, Inc., USA.

Dang, V. D. and Jennings, N. R. (2004). Generating coalition structures with finite bound from the optimal guarantees. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 564–571.

Farinelli, A., Grisetti, G., Iocchi, L., Cascio, S. L., and Nardi, D. (2003). Design and evaluation of multi agent systems for rescue operations. In *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 3138–3143.

Farinelli, A., Iocchi, L., Nardi, D., and Ziparo, V. A. (2006). Assignment of dynamically perceived tasks by token passing in multirobot systems. *Proceedings of the IEEE*, 94(7):1271 –1288.

Farinelli, A., Rogers, A., Petcu, A., and Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 639–646.

Ferreira, Jr., P. R., dos Santos, F., Bazzan, A. L. C., Epstein, D., and Waskow, S. J. (2010). RoboCup Rescue as multiagent task allocation among teams: experiments with task interdependencies. *Journal of Autonomous Agents and Multi-Agent Systems*, 20(3):421–443.

Florek, K., Łukaszewicz, J., Perkal, J., Steinhaus, H., and Zubrzycki, S. (1951). Sur la liaison et la division des points d'un ensemble fini. In *Colloquium Mathematicae*, volume 2, pages 282–285. Institute of Mathematics Polish Academy of Sciences.

Gaston, M. E. and des Jardins, M. (2005). Agent-organized networks for dynamic team formation. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 230–237.

Göbelbecker, M. and Dornhege, C. (2009). Realistic cities in simulated environments – an open street map to RoboCup Rescue converter. In *4th Int'l Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster (SRMED 2009)*, Graz, Austria.

Goldman, C. V. and Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174.

Gunn, T. and Anderson, J. (2013). Effective task allocation for evolving multi-robot teams in dangerous environments. In *Int'l Joint Conf. on Web Intelligence and Intelligent Agent Technologies (WI/IAT)*, volume 2, pages 231–238.

Harriott, C. E., Zhang, T., and Adams, J. A. (2011). Evaluating the applicability of current models of workload to peer-based human-robot teams. In *ACM/IEEE Int'l Conf. on Human-Robot Interaction (HRI)*, pages 45–52.

Kawatsuma, S., Fukushima, M., and Okada, T. (2012). Emergency response by robots to fukushimadaiichi accident: summary and lessons learned. *Industrial Robot*, 39(5):428–435.

Kincaid, J. P., Donovan, J., and Pettit, B. (2003). Simulation techniques for training emergency response. *International Journal of Emergency Management*, 1(3):238–246.

Kitano, H. and Tadokoro, S. (2001). RoboCup rescue: A grand challenge for multiagent and intelligent systems. *AI Magazine*, 22(1):39–52.

Kleiner, A., Farinelli, A., Ramchurn, S., Shi, B., Maffioletti, F., , and Reffato, R. (2013). RMASBench: Benchmarking dynamic multi-agent coordination in urban search and rescue. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1195–1196.

Korsah, G. A., Stentz, A., and Dias, M. B. (2013). A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512.

Kruijff, G.-J., Tretyakov, V., Linder, T., Pirri, F., Gianni, M., Papadakis, P., Pizzoli, M., Sinha, A., Pianese, E., Corrao, S., Priori, F., Febrini, S., and Angeletti, S. (2012). Rescue robots at earthquake-hit Mirandola, Italy: A field report. In *IEEE Int. Symp. on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–8.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2(1):83–97.

Liemhetcharat, S. and Veloso, M. (2012). Modeling and learning synergy for team formation with heterogeneous agents. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*.

Liu, L. and Shell, D. (2012a). A distributable and computation-flexible assignment algorithm: From local task swapping to global optimality. In *Proc. Robotics: Science and Systems (RSS)*.

Liu, L. and Shell, D. A. (2012b). Large-scale multi-robot task allocation via dynamic partitioning and distribution. *Autonomous Robots*, 33(3):291–307.

Manisterski, E., David, E., Kraus, S., and Jennings, N. R. (2006). Forming efficient agent groups for completing complex tasks. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 834–841.

Matignon, L., Jeanpierre, L., and Mouaddib, A.-I. (2012). Distributed value functions for multi-robot exploration. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 1544 –1550.

Murphy, R. R. (2014). *Disaster Robotics*. MIT Press.

Murphy, R. R., Tadokoro, S., Nardi, D., Jacoff, A., Fiorini, P., Choset, H., and Erkmen, A. M. (2008). Search and rescue robotics. In Siciliano, B. and Oussama, K., editors, *Springer Handbook of Robotics*, pages 1151–1173. Springer Verlag.

Nanjanath, M., Erlandson, A., Andrist, S., Ragipindi, A., Mohammed, A., Sharma, A., and Gini, M. (2010). Decision and coordination strategies for RoboCup Rescue agents. In *Proc. SIMPAR*, pages 473–484.

Parker, J., Godoy, J., Groves, W., and Gini, M. (2014). Issues with methods for scoring competitors in robocup rescue. In *Autonomous Robots and Multirobot Systems Workshop at AAMAS*.

Pujol-Gonzalez, M., Cerquides, J., Farinelli, A., Meseguer, P., and Rodríguez-Aguilar, J. A. (2014). Binary max-sum for multi-team task allocation in robocup rescue. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, Paris, France.

Ramchurn, S., Farinelli, A., Macarthur, K., Polukarov, M., and Jennings, N. (2010a). Decentralised coordination in RoboCup Rescue. *The Computer Journal*, 53(9):1–15.

Ramchurn, S., Polukarov, M., Farinelli, A., Jennings, N., and Trong, C. (2010b). Coalition formation with spatial and temporal constraints. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1181–1188.

Rathod, P. and des Jardins, M. (2004). Stable team formation among self-interested agents. In *AAAI Workshop on Forming and Maintaing Coalitions in Adaptive Multiagent Systems*.

Rosenschein, J. S. and Zlotkin, G. (1994). Consenting agents: Designing conventions for automated negotiation. *AI Magazine*, 15(3):29–46.

Sandholm, T., Larson, K., Andersson, M., Shehory, O., and Tohmé, F. (1999). Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111(1–2):209–238.

Scerri, P., Farinelli, A., Okamoto, S., and Tambe, M. (2005). Allocating tasks in extreme teams. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 727–734.

Sedaghat, M. N., Nejad, L. P., Iravanian, S., and Rafiee, E. (2006). Task allocation for the police force agents in RoboCup Rescue simulation. In *LNAI*, volume 4020, pages 656–664. Springer.

Service, T. C. and Adams, J. A. (2011). Coalition formation for task allocation: Theory and algorithms. *Journal of Autonomous Agents and Multi-Agent Systems*, 22(2):225–248.

Shehory, O. and Kraus, S. (1998). Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200.

Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems. Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, Cambridge, UK.

Siddhartha, H. (2009). Retrospective analysis of RoboCup Rescue simulation agent teams. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1365–1366.

Skinner, C. and Ramchurn, S. D. (2010). The RoboCup Rescue simulation platform. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 1647–1648.

Stone, P., Kaminka, G. A., Kraus, S., and Rosenschein, J. S. (2010). Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proc. Nat'l Conf. on Artificial Intelligence*.

Trăchioiu, M. (2014). Hierarchical decision theoretic planning for robocup rescue agent simulation. Master's thesis, Universiteit van Amsterdam.

van Santen, W., Jonker, C. M., and Wijngaards, N. (2009). Crisis decision making through a shared integrative negotiation mental model. *Int'l Journal of Emergency Management*, 6(3/4):342–355.

Vig, L. and Adams, J. A. (2007). Coalition formation: From software agents to robots. *Journal of Intelligent Robotic Systems*, 50(1):85–118.

Waugh, W. L. and Streib, G. (2006). Collaboration and leadership for effective emergency management. *Public Administration Review*, 66:131–140.

Xu, Y., Scerri, P., Yu, B., Okamoto, S., Lewis, M., and Sycara, K. (2005). An integrated token-based algorithm for scalable coordination. In *Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 407–414.

Zavlanos, M. M., Spesivtsev, L., and Pappas, G. J. (2008). A distributed auction algorithm for the assignment problem. In *Proc. 47th IEEE Conference on Decision and Control*, pages 1212–1217.

Zhang, Y. and Parker, L. E. (2012). Task allocation with executable coalitions in multirobot tasks. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 3307–3314.

Zhang, Y. and Parker, L. E. (2013). Considering inter-task resource constraints in task allocation. *Journal of Autonomous Agents and Multi-Agent Systems*, 26:389–419.

Zheng, X. and Koenig, S. (2008). Reaction functions for task allocation to cooperative agents. In *Proc. Int'l Conf. on Autonomous Agents and Multi-Agent Systems*, pages 559–566.

Zheng, X. and Koenig, S. (2009). K-swaps: cooperative negotiation for solving task-allocation problems. In *Proc. Int'l Joint Conf. on Artificial Intelligence*, pages 373–378.