UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering, Science and Mathematics

School of Electronics and Computer Science

Coalition Formation in RoboCup Rescue with Spatial and Temporal Constraints

by

Chrysovalandis Agathangelou

September 22, 2011

A dissertation submitted in partial fulfillment of the degree of
MSc Web Technology
by examination and dissertation

Supervisor: Dr. Sarvapali Ramchurn
Examiner: Dr Enrico H Gerding

Abstract

Every day catastrophic disasters are happening around the globe, most of the times are small, however sometimes they may have tremendous size, like earthquakes, hurricanes etc. After such disasters rescue forces have to save as many injured civilians as possible and clean up the mess. Rescue forces may have limited communications, many roads may be blocked and some fires may be spreading. It is critical for the rescue forces to be able to cooperate and choose the most efficient plan, taking into considerations the limitations they have to face, for maximising the number of saved people. This thesis focus on studying about the RoboCup Rescue project and the state of the art approaches based on their categories and their performance. In addition, the problem of planning the moves of the rescue force, is modelled in a novel way and a new algorithm is proposed. Furthermore, the introduced algorithm is evaluated by theoretically proving some of its properties against the F-MAX-SUM, a state of the art algorithm. Finally, some potentially good ideas for the future are discussed.

Contents

List of Algorithms

1 The RoboCup Rescue Project

After some catastrophic earthquakes with the most important in Kobe city, Japan and after the success of the RoboCup Soccer project [10] the RoboCup Rescue project was initiated in 1999[11],[9]. Its purpose was to promote research in the area of multi-agent systems, information systems and other areas related with search and rescue, in a way that rescue forces, with intelligent and efficient planning will maximize the number of saved civilians. Before this project there wasn't any simulator that could contribute to the development of better methods for rescue forces. In addition heterogeneous agents can benefit from their differences and use the unique abilities of each type of agent to maximize the effectiveness of the rescue force. The RoboCup rescue project is organized in 4 sub-projects [8] : simulation, robotics and infrastructure, integration and operation. The purpose of simulation projects is to develop disaster simulators along with decision systems for agents that can plan the best possible strategy in order to save the maximum number of injured people possible. The robotics and infrastructure project aims to build up robotic equipment that can be used in real life destructions, establish digital assistants and evolve information infrastructures to be able to support the efforts of the rescue force. The integration project's objective is to combine and deploy simulation and robotics projects together. The operation project's goal is to utilize the system in an effective way in real life destructions.

2 Introduction

After a disaster, rescue forces have to coordinate in order to save as many people as possible and to clean up the mess. People are buried, roads are blocked and buildings are burning. Rescue forces consist of ambulances that save people and bring them to hospital, fire brigades that extinguish fires and the police force that is responsible to open the roads. The agents can communicate with other agents by exchanging messages even if they are different types of agents (e.g. police brigade with ambulance).

Tasks that agents have to perform involve rescuing civilians, extinguishing fires and clearing roads. Each task has a deadline, for example the time that a specific trapped civilian can survive without help, or the time that a fire needs to destroy a building. Also tasks have a workload, which represents the number of time units of work that are necessary in order for the task to be completed. Tasks

are spatially distributed, and agents travel with a standard predefined speed. So in order to move from one task, after completion, to a new task agents need to spend some time to travel. This is the reason that agents must carefully take into consideration the topology of each task in order to be routed in a way that maximizes the number of rescued civilians.

Some tasks need a large amount of resources to be completed; therefore more agents may have to coordinate in order to complete such tasks. This is the reason that agents form coalitions. When cooperating, agents may need less time to complete a task rather than the sum of the time needed by each agent to accomplish the same task alone.

"The value of a coalition" represents how well agents, in a coalition, can cooperate and determines the amount of time a coalition needs to complete a task. For example if agents A and B cooperate to do task T they may need one time unit. But for A or B to complete the task alone may take three time units for each agent, in order to complete the same task. Hence by cooperating they minimize the time to get the utility of a task by completing it. Different sets of agents can have different synergistic abilities. "The value of a coalition" is important, because in the real world two agents can perform better than one. For example if a building is in fire and two fire brigades spray water in the fire from different angles, this may help extinguish the fire faster rather than throwing water from the same angle. After a task is completed or after a new task is added, coalitions may disband and new coalitions are formed based on tasks left. It is important for an agent to decide which coalition is the most optimal to join and with which task a coalition shall engage in a way that more people can be saved. In addition agents have to face a number of problems that make the problem significantly more difficult. After a disaster, communication centers, antennas and other infrastructure may be destroyed causing the communication network to collapse. Also the smoke derived from fires, some other physical obstacles and the limited battery of radios may make it difficult for agents to communicate with other agents. Hence we assume that agents have limited communication capabilities. Furthermore the environment is considered to be dynamic. New tasks may appear and old tasks may disappear all the time, therefore agents need to adjust and consider this in their strategy. For this reason the algorithm that solves the problem must be fast so that agents can be able to re-run it often in order to consider up to date information. Moreover the resources of the rescue force are limited due to the fact that the number of tasks is much larger than the number of agents. Finally we cannot assume the existence of a central decision point that controls the agents

because of the communication problems caused by the disaster. For this reason a distributed algorithm must be used that also has the advantage of no single failure point. Also this project is mainly focused in saving civilians in contradiction with other projects that also consider fires and blocked roads as tasks. Extra types of tasks can be considered in future work.

The rest of this paper is organised as follows. In Chapter 4 some state of the art approaches, that are related with the RoboCup Rescue problem, are categorised in centralised, decentralised and coalition formation approaches and discussed. In Chapter 5 we discuss the model that will be used in this paper. After that Chapter 6 defines some properties that are essential for the DARRC algorithm. Next on Chapter 7 the DARRC algorithm is described and some key points of the algorithm are discussed. In Chapter 8 some properties of the DARRC algorithm are proved and in Chapter 9 the proposed algorithm is evaluated. Finally in Chapter 10 some ideas that can be useful in future work are discussed and Chapter 11 concludes.

3 Literature Review

Most of the work done up to date for RoboCup rescue problem is focused on how to achieve better score on the RoboCup Rescue competition. To do so they exploit bugs of the simulator or the proposed algorithms are designed to work only in specific environments (i.e. graph with high density of tasks) and do not try to formalize the problem in a way that it can solve real life incidents. However some methods that were designed to solve this problem are different and could probably perform or be enhanced in order to perform well enough in real life disasters. In what follows some of the proposed methods that are considered state of the art are examined. Firstly, two centralised approaches are described, a method that uses neural networks and a method that uses reaction functions. Second, two decentralised approaches are presented, one that uses Markov games and the Max-Sum algorithm. Third, three approaches that focus on coalition formation are examined. Finally, some methods that can be useful in this area, by providing methods of communication, creating map representation, clustering etc are illustrated.

3.1 Cetralised Approaches

Centralised approaches were the first that were examined in the area of RoboCup Rescue. The reason is the complexity of the problem makes it easier to assume a central planner that controls all the agents of the rescue force. Centralised approaches achieve better results compared to decentralised approaches because of the previous assumption. However in order for a centralised method to work, the central planner must be able to communicate with all the agents, all the time. Also sending all these messages to the agents requires a network with a very large bandwidth. Furthermore the central planner, in order to process all this information and calculate the optimal actions for all the agents, requires a huge amount of processing power. Moreover if the central planner fails, then the whole rescue force will be paralysed. All the injured civilians will be left without help and eventually die and the fires will burn the city. In addition it is almost impossible to communicate with so many agents simultaneously.

3.1.1 Random Neural Networks with Synchronised Interactions

Timotheou et al. proposed a centralised algorithm [7] that uses Neural Networks to solve the problem of assigning rescue agents to tasks in an optimal way. In order to train the network some recorded incidents are used. The optimal solutions of these incidents are calculated off-line and then are given as input and expected result on the neural network respectively. After that, some random incidents are given as input to the network in order to test it. Two network architectures with four different approaches were tested showing that this approach is effective in giving approximate solutions and can produce solutions relatively fast which is important for the performance of the algorithm in real life situations. However the scenario that this method solves is simplified and ignores some aspects that are important for a successful rescue mission such as the cooperation between agents by forming coalitions. It is assumed that only one agent can work on a specific incident at a time. Because of the previous assumption, a unit must have a capacity at least equal to the number of the people that are trapped or injured in a task in order to be able to engage with that task.

3.1.2 Reaction Functions for Task Allocation to Cooperative Agents

Zheng et al. developed a decentralised algorithm [19] that solves the RoboCup Rescue problem by using reaction functions to allocate tasks to available agents. It is assumed that all agents are of the same type (e.g. fire brigades that can also transfer injured civilians to hospitals) and the locations of all tasks are known. Also it considered that a central planner exists and can communicate with all agents. Additionally a task must be visited simultaneously by a predefined number of agents, which can be different, depending on each task, hence coalitions must be formed. A coalition is defined as the number of agents that are necessary to complete a specific task. Zheng et al. developed a method called reaction functions which is "a novel way of characterizing the costs of agents in a distributed way". A central planner assigns tasks that only need one agent's work in order to be completed by greedily calculating what the best choice is for the rescue force as a whole. The central planner makes the decision as if it is the last move of the agent. After assigning all the single tasks, one at a time, the central planner assigns agents to coalitions with sequential single item auctions. Each agent in a coalition has a commitment to be at a specific location in a predefined time. Simple tasks are assigned first so that each agent can change the order of visiting the targets in a way that coalitions are favored and the global utility is maximised. This method has achieved very good results. However, it is assumed that a central planner exists and all agents can communicate with the central planner. This is a problem because after a disaster it can be impossible to achieve communication between all the agents. Also this algorithm needs a lot computational time (especially from the central planner) and has great communication overhead since all agents have to communicate repeatedly with the central planner.

3.2 Decentralised Approaches

Nowadays decentralised approaches are more popular because they do not have the limitations of centralised approaches (e.g., global communication, computation overhead, network overhead and single point failure). However, despite the advantage of not having these limitations, it is very difficult to develop a distributed algorithm and therefore most decentralised algorithms that were proposed up to date are greedy. Also distributed algorithms are less efficient compared to the centralised algorithms mentioned earlier.

3.2.1 Decentralised dynamic Task allocation: a practical game-theoretic approach

Chapman et al. proposed a decentralised algorithm (OPGA) [3] for allocating tasks, by using Markov game formulation. Here it is assumed that there is only one type of rescue forces, i.e., ambulances and hence all agents are ambulances. Each task (i.e., a person to be saved) has a deadline and a predefined time needed in order to be completed. Some tasks may require more than one agent in order to be completed because the effort needed to complete the task is too big and the deadline of the same task is too early for an agent to reach it alone. Also a task cannot be divided in subtasks. Each agent has a utility function that is related only to it and it aims to maximise it. In order to force all agents to contribute in global welfare, when the global utility changes then the private payoff of each individual agent changes the same way that the global payoff changes (i.e., both global and private utility increase or they both decrease). For every time step, one game is constructed simulating the strategy of the other agents for a predefined number of future steps. At each time step, the Markov game is approximated with a static game that is assumed to have complete information about the environment. This method was tested against 2 greedy algorithms in 3 different maps, each time having to rescue 80 civilians with only 6 ambulances. Each map differs in the size and how fast new tasks appear. Most of the time the centralised greedy algorithm performs better than OPGA and the decentralised greedy algorithm performs worse than OPGA. OPGA performs almost the same with the centralised algorithm in the map that is bigger and has less structure than the other maps. However, in the map where incidents appear more rapidly than the other maps, as expected, centralised greedy algorithm performs much better than OPGA. This is mainly because a decentralised algorithm saves time with fast decisions and due to the fact that the extra time steps that OPGA needs in order to calculate future movements are useless because the environment changes quickly. Also decentralised solutions are faster than centralised solutions, because they solve the problem in parallel. In contrast, when the range of communication between the agents is limited, OPGA outperforms both greedy algorithms. This is due to the fact that OPGA benefits from the use of Markov models in comparison with the myopic approach of the greedy algorithms that make their decisions considering only the short term future. An important observation is that Markov model helps in cases where limited communication can be achieved and agents most of the time don't know the intentions of the other agents. Another interesting conclusion derived from this paper, is that when the environment is highly dynamic, plans should be

made only for the short term future.

3.2.2 Decentralized Coordination of Low-Power Embedded Devices using the Max-sum Algorithm

The Max-Sum(MS) algorithm [5] was designed to solve the problem of maximising social welfare in groups of agents that can interact with each other. Each agent is represented in a cyclic bipartite factor graph, with variable nodes representing agents state and function nodes representing agents' utilities. The utility of an agent depends on its own state and the states of other agents adjacent to it. The goal of this algorithm is to choose the state of each agent in a way that social welfare is maximised. Furthermore a way of passing messages in a cyclic graph is provided in this paper. The proposed algorithm was compared with Distributed Stochastic Algorithm (DSA), Best Response (BR) and Dynamic Programming Optimisation Protocol (DPOP). The algorithms were tested in solving a set of instances of graph coloring problems. Tested algorithms were evaluated based on the number of conflicts over time, exchanged messages size and number. The algorithm was left to run for a predefined number of cycles; nevertheless there are cases when this is not enough. Generally Max-Sum has fewer conflicts over time compared with the other 2 approximate algorithms. However the messages that MS sends are significantly bigger than the other 2 approximate algorithms but smaller than DPOP. Also MS tends to send more messages than the other approximate algorithms. However by sending more messages MS is more robust than the other algorithms in case of noise in communications. Moreover MS was used by other methods that take into consideration more aspects of real world destructions, like the fact that agents can benefit from creating coalitions.

3.3 Coalition Formation

The main difference of methods that belong in this category, compared with other approaches is that they try to exploit The value of a coalition. To do so, some of the methods invented ways to search and find the best possible coalition for a specific task, from all possible coalitions. However, this extra variable complicates more the RoboCup Rescue problem, making it extremely difficult to solve. Also most of the approaches on this category are distributed.

3.3.1 Decentralized Coordination in RoboCup Rescue

The algorithm that was proposed by Ramchurn et al. [15] is different from previous approaches because they model the same problem but with coalition formation and in a decentralized fashion. Initially the agents have knowledge only of the map of the area, tasks are unknown and they have to search to find tasks. In this paper optimal and approximate solutions of the problem are defined taking into consideration that coalitions are formed. The introduced decentralized algorithm is based on Max Sum Algorithm [5]. It can complete 10% more tasks than the state of art decentralized algorithms. Also it needs less computation time and number of messages exchanged compared to the Max Sum algorithm. Their algorithm is greedy because agents consider only one task at a time, in order to make the algorithm faster. Also some heuristics are used in order to maximize the number of tasks that can be done in minimum time and hence to be able to complete other tasks in the future. One simple but useful heuristic is that the algorithm does not take into consideration tasks that are impossible to complete before their deadline. In this fashion the number of tasks that have to be scheduled is reduced hence less computational time is needed to run the algorithm. Furthermore time and resources are no longer wasted on tasks that are impossible to complete and therefore no utility would be gained from them. Another heuristic is that the algorithm focuses on sending the agents with more capabilities to the most demanding tasks. This way precious time is saved because fewer vehicles have to move for a specific task.

3.3.2 Coalition Formation with Spatial and Temporal Constraints

In this paper [16] a new method based on mixed integer programming is introduced in order to solve the coalition formation problem in RoboCup rescue. It takes into consideration that an agent may work more effectively with some agents compared to when working with some other agents. This assumption further complicates the problem but it simulates the real life synergistic capabilities that units of the rescue force have. Also a major improvement over previous work is that if an agent or a set of agents work on a task for a number of time units, then the workload of the task decreases linearly depending on the total capacity of the agents that work on it. The mixed integer programming method optimally solves the problem. However, it needs exponential time to be solved and hence it is impossible to scale to a bigger number of agents or tasks. Nevertheless this method is important because it can be used to compare other proposed solutions with the

optimal solution. Furthermore some brilliant heuristics that can help speed up the solution computation are proposed. The coalition problem is solved after deciding which is the smallest number x of agents that are needed to complete a task by its deadline and then a coalition is chosen, from other coalitions that consist from x agents, greedy. The proposed method was tested with simulations of 2 to 20 agents that have to accomplish 300 tasks and compared with EDF [13] algorithm. The proposed method outperforms EDF most of the times by completing up to 31% more tasks.

3.3.3 Distributed coordination architecture for multi-robot formation control

Most approaches for coordination between agents are leader-follower based. In this article [17] a hybrid approach, of leader-follower and consensus algorithm, is proposed. This approach exploits the topology of the agents that, most of the times, form small neighbourhoods and a leader is assigned in each neighbourhood. The leaders coordinate, using the consensus algorithm, in order to lead their neighbourhoods to become more efficient with regard to the global utility, not only the local. Additionally the members of each neighbourhood, apart from the leader, can communicate only with members of the same neighbourhood. In this way fewer messages are send and agents cooperate better.

3.4 Other Related or Useful Work

Papers that belong in this category are considered to be important work that could be used by algorithms in the area of the RoboCup rescue problem. However most of these papers are not related with the main algorithm (the one that plans the next moves) but have to do with other issues, like map representation, creating ad-hoc networks by the rescue force during a disaster, etc. Even though they are not used by the main algorithm this work is important because small improvements related with the secondary methods (e.g., mapping, clustering, networking) can significantly improve the overall performance of the algorithm.

3.4.1 Evaluating the RoboCup 2009 Virtual Robot Rescue Competition

In [2] the key aspects of performance evaluation, in the RoboCup virtual competition of 2009, are analysed. The competition consists of three challenges. The first challenge is the mapping challenge, where the participating teams have to map an indoor or outdoor environment in 20 minutes using up to four robots. The purpose of this challenge is for robots to be able to find the best route in the environment that was mapped in case of an emergency. The robots that are responsible for the mapping have to communicate with each other in order to achieve their goal, despite of the artificially added noise in the communications. The second challenge is the Deployment challenge where agents have to achieve communication coverage of as much square meters as possible. Agents have maps of the building, but some paths are blocked due to destruction and agents have to be able to communicate with each other. Finally the third challenge of the competition is the Teleoperating challenge, where 8 robots must reach 8 different locations. Robots are different and hence are better for different targets, but do not know which robot is better for which task.

3.4.2 A Scalable Hybrid Multi-Robot SLAM Method for Highly Detailed Maps

With the developments that took place in the last few years in robotics it became vital, in competitions or in order to improve, for robotic agents to be able to create an accurate representation of a map. With a good map representation, an agent can constantly know its position and other information that can help it make a better decision. Also the programmers of the agent can look at the produced representation and improve the design of the agent. The SLAM method presented in [12] is a method for map representation that was proposed by Pfingsthorn et al.This method in contrast with other, state of the art, methods is scalable and has similar accuracy. Furthermore in the RoboCup World Championships of 2006 [1] the SLAM method of Pfingsthorn et al. was nominated with the “Best Mapping Award”.

3.4.3 Combining Exploration and Ad-Hoc Networking in RoboCup Rescue

Teams of robots are used to explore an unknown environment and they have to maintain communication with a central station. The purpose of these robots is to send information back to the central station, regarding the location or other information about victims or building on fire. Some robots can act as relays between another robot and the central station, hence creating an ad-hoc network. The purpose of the proposed method in [18] is to maximize the utility by maintaining communication or explore new territories. Agents have to balance between these two tasks in order to maximize the global utility. The algorithm was tested with simulations. The metric used in the tests was the percentage new territory was explored over the maximum possible new territory to be explored. With up to 2 robots the algorithm covers almost 100% of the maximum possible, however with more robots this algorithm does not perform so well.

3.4.4 Clustering by passing messages between data points

Most current methods for clustering choose randomly a set of data points and then they try to improve this set of points. However, the results of these methods are strictly connected with the quality of the initial set of data points chosen. Since this choice is random there is a big probability to produce bad results. The proposed method [6], instead of choosing randomly a set of data points it considers all of them as potential exemplars, and through a process a better set of exemplars is chosen. The proposed algorithm takes as input a graph and the physical similarity between the nodes of the graph. Also it is not necessary to predefine the number of clusters. Two types of messages are exchanged between the nodes. The first type represents of how well a point x can represent another point y. The second type represents the level of support a node gets, to be included in the exemplar, from other nodes. This method was compared with k-centers clustering algorithm in finding exemplars in images and achieved less error compared to the k-centers algorithm. Also, the affinity propagation based method needed significantly less processing time in order to reach convergence. In the test of identification of gene clusters affinity propagation again had more accuracy and was about 2000 times faster than the k-centers algorithm. Affinity propagation also outperformed the k-centers algorithm in more tests.

3.4.5 ALADDIN Rescue team description

ALADDIN Rescue team have used techniques that were used by previous winners of the competition. Also they designed an algorithm in a way that police force, fire brigades and ambulances can cooperate better and act as a team. In this paper [14] Bayesian techniques are used to estimate how much the priority of discovering new tasks based on the type of event and knowledge of a predefined number of events. The ambulances use heuristics in order to determine the order that they will visit the tasks. For example if 2 tasks are candidates for the next step, then the task with the earliest deadline is selected first. Police force, which is responsible to open the road, decides which road to open based on the priority of the type of the task that the blocked road is an obstacle for rescue forces. For examples blocked roads that lead to trapped civilians have higher priority than roads which lead to fires. Also a way to calculate the priority of each blocked road is to compare the utility that will be lost if this road is not opened. When an agent of police force is allocated to a task, the priority of the task becomes less. If police have no priority task left, then they monitor the city and constantly inform the other agents about the situation. In addition, fire brigades decide which fire to extinguish based on the neighbor buildings, in order to prevent fire from destroying more buildings and continue burning. Also fires that are near building that have civilians have higher priorities compared to other fires. In order to achieve that a fire simulator was developed which, based on available information, calculates how the fire will spread in the future.

3.5 Summary

The work above was studied and presented in this paper because it is considered essential for the area of RoboCup Rescue. Some ideas were used in the proposed algorithm and some can be used in the future.

Between the centralised approaches the method that uses neural networks is the most promising. Despite the fact that neural networks are a very old approach, this method is very good in finding approximate solutions. The proposed method by Timotheou et al. achieved very good results and can be extended in order to be used in real life disasters. Also a distributed algorithm based on Neural Networks could be created, using the same concepts that were used in this method. The method that uses reaction functions also achieves good results but is not as promising as Neural Networks because there is not much room for improvement.

However the concept of using sequential single item auctions is brilliant and can be used in distributed methods. Both of these methods are very difficult to be used in real life disasters, because of their centralised nature that has some serious limitations that were explained earlier.

Among the decentralised approaches, OPGA even though that was outperformed by a centralised approach for most of the cases and in some of the cases was outperformed even by a greedy algorithm, has potential to become better. The idea of using Markov game formulation can be combined with other methods to predict the next move of other agents and improve the performance of the algorithm. Also [3] contributes because of finding the cases that OPGA is better and the cases that decentralised and centralised approaches are better and explaining the reason of these results. Max sum is important because it provides a way of maximising social welfare and it is used in other algorithms (i.e. F Max Sum).

Regarding the coalition based approaches, F-Max-Sum algorithm [15] is important because it is one of the first efficient algorithms that exploit the ability to form coalition. It improves Max-Sum algorithm results by taking into consideration the Coalition Value. Also some useful heuristics were proposed and can be used in the future by other methods. The Mixed Integer Programming (MIP) method improves F-Max-Sum algorithm in some cases. It is very important because it uses MIP in order to solve the RoboCup Rescue problem. Furthermore some new clever heuristics were introduced which could also be used in the future. Moreover, the last approach [17] was studied because of the novelty of the method used in order to categorise agents into neighborhoods and assign leaders to each neighborhood.

Additionally, the paper that evaluates a virtual RoboCup Rescue competition belongs to the category of other related or useful work. It is considered important because it offers an alternative simulator from the original of the RoboCup Rescue project and evaluates the performance of the proposed algorithms based on different metrics. It is important to have at least two different benchmarks for these algorithms, because some may exploit bugs in one of the two benchmarks, or the benchmarks can focus on different areas that are both considered important in order to improve overall performance. By evaluating an algorithm, with more than one benchmarks, will motivate the developers to further improve their algorithms, in contrast with some cases that an algorithm performs very well when using the RoboCup rescue simulator and the developers consider their work done. The SLAM method for mapping is vital for agents because it allow them to monitor changes on the map that were caused by a catastrophe and navigate inside

buildings that no maps are available. SLAM was nominated with the best mapping award of the 2006 RoboCup Rescue Championship, which guaranties its high level of performance. Furthermore the Ad-Hoc method can be used by agents in order to maintain communication with the maximum possible number of agents and by using fewer messages in order to maintain communication. ALADDIN Rescue team was considered important to mentioned because of the work that this team produced(simulators, new methods) and the awards won, which shows how significant was the work produced. In [14] some good ideas are presented, like using Bayesian techniques in order to determine whether to move for existing tasks or discover new tasks. In addition, the proposed method for finding which road to use could be useful in the future.

The DARRC algorithm was decided to be distributed, because of the advantages of distributed algorithms over centralised that were described earlier and due to the fact that a distributed approach can be implemented easily in real world. Also F-Max-Sum algorithm was decided to be used in the proposed algorithm because it has higher performance overall if compared with other similar algorithms. Moreover, some heuristics from the mix integer algorithm were used because they can improve performance. The rest of the parts that were examined in literature search can be used in the future. Finally, the clustering algorithm was preferred over other Neural Network approaches, that achieved very good results in the past, because it has slightly better results and sometimes it can be 200 times faster than NN approaches, which is vital in RoboCup Rescue.

4 Model

The model that is used in this thesis is similar to the model of [15]. Agents are represented with $a_1, a_2, \ldots, a_n$ and the set of all agents is represented with A. If $i < j$ then agent a_i has bigger priority than agent a_j. The tasks that have to be completed are noted with $t_1, t_2, \ldots, t_n$ and all tasks belong to the set T. Each agent has a capacity that can be found by using the function $c : A \rightarrow [0, \infty]$. The Capacity of an agent defines how fast an agent can work. It represents the units of work an agent can perform per time unit. Every task has a deadline given by the function $d : T \rightarrow [0, \infty]$, a workload $w : T \rightarrow [0, \infty]$ and a utility $u : T \rightarrow [0, \infty]$. The deadline of all tasks is the same. In real life this can be interpreted, that after a destruction a second destruction may happen. For example after an earthquake a huge wave may strike after a small amount of time.

Therefore agents with as many injured civilians as possible must return to a shelter before the second event. Also there are other cases that the notion of the same deadline may be useful, such as bombing of a place during a war, the time of the attack my be known. The workload of a task reduces depending on how much work some agents have performed on that task. If agent a_i works on task t_i, the workload $w : (t_i)$ at time $\tau + x$ equals $w(t_i)$ at time τ minus $c(a_i) \times x$. In order for a task to be completed successfully, its workload must be equal to 0 before its deadline. If a task is successful, its utility is added to the global utility. Utility of each task is given as input and hence it does not necessarily have any relation with the workload of the same task. In addition, utility does not decreases over time. The tasks or agents are placed in any location that belong to L, a set of all the possible locations of the map that an agent or a task is possible to be placed. Each location L_i represents a point in the map, having coordinates (x_i, y_i). Together x_i and y_i can uniquely denote a location, L_i. $L_i = L_j$ iff $x_i = x_j$ and $y_i = y_j$. All the agents are considered to have the same speed. A distance between two locations is given by the function $\rho : L \times L \rightarrow [0, \infty]$. The result that function ρ returns represents the distance between two locations if travelling through the shortest path using only roads. It does not represent the Euclidean distance or radius of the two locations. If $L_i = L_j$ then distance between these two locations is 0. Agents can form coalitions in order to complete a task and after the task is completed the coalition can be disbanded. $C \in 2^A$ is the set of all possible coalitions that agents can form anytime. An agent that works on a task at a specific time $\tau \in [0, \infty]$ is denoted as $\tau^{A \rightarrow T}$, in the same fashion when a coalition works with a task is represented as $\tau^{C \rightarrow T}$. All agents that belong to a coalition that works with a specific task must be in the place that the task is located in the same time. However if some agents of a coalition arrive at the location of a task before some other agents of the same coalition, then they can start working on the task. It is assumed that all coalitions have the same coalition value and that coalition value is not related with the number of agents that consist the coalition. Hence the total capacity of any coalition will be the sum of the capacity of its agents. An agent that belongs to a coalition can be the leader of the coalition θ_l or just a follower θ_f. However if an agent was chosen not to be in a coalition is denoted with θ_n. Each coalition has only one leader. Simple tasks are the tasks that can be completed before the deadline by a single agent. Complex tasks are the tasks that need more than one agent to work on them, in order to be completed before deadline. New tasks can be added at any point of the time at any possible location $L_i \in L$. This property simulates real life situations, when distress calls add, or agents discover new, tasks. The rate of new tasks defines how many new

tasks are discovered or added per time unit. The communication range of the agents can vary. On some occasions agents may be able to communicate with all other agents that are placed in any possible location that belongs to L. However there are cases where agents can communicate only with agents that are placed in locations that belong to a small subset of L. The function $comms(a_i, a_j)$ returns 1 if agent a_i can communicate with agent a_j. Else $comms()$ function returns 0. The purpose of the agents is to achieve as much global utility as possible. The whole model can be represented as a graph $G = (V, E)$ having as nodes all the tasks of the set T ($V = T$) and as edges the shortest distance to their neighbor tasks. If the shortest distance from a node, x to another node y is by traveling through node z, then the edge xy is not included in the graph. Only the edges xz and zy are included in the graph, hence $(x, z) \in E$ and $(z, y) \in E$. E is the set of all edges of the graph that alternatively represents the model.

5 Properties of the Problem

In this section some useful statistics that are necessary to be calculated later in the proposed algorithm are explained. The averages of workload, capacity, distance, reach time, deadline, maximum work, total tasks, utility and utility over time are defined. Furthermore total capacity, capability factor and total workload of complex tasks are specified.

- AverageWorkload

The sum of the workload of all the tasks divided by the number of the tasks. It is useful in calculating the $AverageMaxWork$ property.

$AverageWorkload = \frac{\sum_{i=1}^{n} w(i)}{n}$

where n is the number of all the tasks.

- Total Utility

The sum of the Utility of all the tasks. It is used in algorithm 2,3 and 6. Using this property the DARRC algorithm calculates the TotalUtility of all the tasks, that the algorithm has planned to complete before the deadline. By calculating the TotalUtility of the path and along with the variable UtilityPer, DARRC can

compare it to the utility that is expected to be gained from the cluster and hence the algorithm can determine if it is a good choice to travel to the cluster or if it is better to run the Greedy Algorithm.

$TotalUtility = \sum_{i=1}^{n} u(i)$

where n is the number of all the tasks.

- Average Capacity

The sum of the capacity of all the agents divided by the number of the agents. It is used in the process of calculating $AverageMaxWork$ property.

$AverageCapacity = \frac{\sum_{i=1}^{n} c(i)}{n}$

where n is the number of all the agents.

- Average Distance

It represents the average distance between adjacent tasks. The sum of the length of the edges of the graph that connect adjacent tasks divided by the number of the edges. It is useful in calculating the $AverageReachTime$ property.

$AverageDistance = \frac{\sum_{i=1}^{n} length(i)}{n}$

where n is the number of all the edges.

- Average Reach Time

The average time to reach a task. The average time that an agent needs to travel from a task x to a task y which is adjacent to x. It is useful in calculating both the $AverageUtilityOverTime$ and $AverageMaxWork$ properties.

$AverageReachTime = \frac{AverageDistance}{Speed}$

where $Speed$ is the speed of the agents.

- Average Deadline

The average deadline of the tasks. It is useful in calculating the $AverageMaxWork$ property. However, the deadline in this paper is considered to be the same for all the tasks. This property can be useful in future approaches, where the tasks have different deadlines like in[A]

$$AverageDeadline = \frac{\sum_{i=1}^{n} d(i)}{n}$$

where n is the number of all the tasks.

- Average Max Work

The average number of tasks that an agent is capable of performing. It is useful in calculating the $AverageTotalTasks$ property.

$$AverageMaxWork = \frac{AverageDeadline}{\frac{AverageWorkload}{AverageCapacity} + AverageReachTime}$$

- Average Total Tasks

The expected total number of tasks to be completed before deadline. It is useful in calculating the $CapabilityFactor$ property.

$$AverageTotalTasks = AverageMaxWork \times NumberOfAgents$$

- Capability Factor

The average number of tasks, that is expected to be accomplished before deadline, divided with the number of all the tasks. This property is used in an extended version[A] of DARRC that supports different deadline for the tasks. With the $CababilityFactor$ property the DARRC algorithm is able to determine if there is enough time available to complete tasks with less deadline first and does not consider tasks that have sufficient deadline.

$$CapapilityFactor = \frac{AverageTotalTasks}{NumberOfTasks}$$

- Average Utility

The sum of the utility of all the tasks divided by the number of tasks. It is useful in calculating the $AverageUtilityOverTime$ property.

$$AverageUtility = \frac{\sum_{i=1}^{n} u(i)}{n}$$

where n is the number of all the tasks.

- Average Utility Over Time

The average utility of the tasks over the average time needed to reach and complete a task. It is used in algorithms 2,3 and 6. The purpose of this property is to check if a cluster is a good choice to be visited, because agents that will be located inside that cluster will be able to get much utility per time unit.

$$AverageUtilityOverTime = \frac{AverageUtility}{\frac{AverageWorkload}{AverageCapacity} + AverageReachTime}$$

- Total Workload of Complex Tasks

The sum of the workload of all complex tasks. It is useful in calculating the $AverageWorkloadComplex$ property. It is also useful in the algorithm 4. It is used in order to determine how much capacity, the agents that will be assigned to complex tasks, must have.

$$TotalWorkloadComplex = \sum_{i=1}^{n} w(i)$$

where n is the number of all the complex tasks.

- Average Workload of Complex Tasks

The average workload of the complex tasks. It is useful in algorithm 4. It is used in order to determine how much capacity a coalition must have, in order to be able to complete most of the complex tasks.

$$AverageWorkloadComplex = \frac{\sum_{i=1}^{n} w(i)}{n}$$

where n is the number of all the complex tasks.

6 A Decentralised Algorithm for RoboCup Rescue based on Clusters (DARRC)

The Decentralised Algorithm for RoboCup Rescue based on Clusters(DARRC) is designed for the worse case scenario. For example, having a large number of tasks compared to the number of agents and a very early deadline. The purpose of this algorithm is to perform better compared to state of the art algorithms in such cases and equally good in remaining cases. The choice to design based on the

worse case scenario was taken because no other existing method is designed for such cases, and by performing better in such cases and equally good in the rest cases, overall DARRC can outperform other approaches.

6.1 Definitions

Cluster_Utility is the sum of the utility of all tasks, within the cluster, that are expected to be completed before deadline. A cluster is a place where travelling between tasks that belong to this cluster is considered to be efficient, based on the variable of Cluster_Limit.

Cluster_Limit is a variable which defines the minimum AverageUtility that tasks of a cluster must have in order to be optimal to travel to the specific cluster.

At every cycle that the algorithm runs agents send messages, containing information about tasks that were discovered or completed, to the other agents.

UtilityPer in algorithm 2 line 5 and algorithms 3 and 6 is a variable which defines how much the utility gained from a cluster must be compared to the utility gained from the whole path in order to get to the cluster. For example if the TotalUtility that is expected to be gained from the path to the cluster and from the cluster, before deadline, is 100 and the value of UtilityPer is 0.9, then ClusterUtility must be at least 90 in order for the agent to visit the cluster. With this variable and depending on the environment and the case of disaster, with fine tuning DARRC can outperform the existing approaches for RoboCup Rescue.

Cluster Utility in algorithms 2,3 and 6 is the utility that is expected to be gained from tasks that belong to the cluster.

6.2 Alternative Graph and Dijkstra Algorithm

In order to find a way to travel between two tasks, using the most efficient path (in terms of time spent over utility), it is necessary to construct a new graph, G' which is based on G. Graph G' has the same nodes as the graph G. However, its edges are different and are calculated based on the distance of the edges of graph G, the workload and the utility of the tasks that are represented by the nodes on each edge of the graph G.

For every edge of graph G that connects nodes i and j, two new edges are added with weight calculated as follows.

$$\text{weight of } ij = \frac{\frac{w(j)}{TotalCapacity} + \frac{distance(ij)}{speed}}{u(j)}$$

in the same fashion ji is calculated

$$\text{weight of } ji = \frac{\frac{w(i)}{TotalCapacity} + \frac{distance(ij)}{speed}}{u(i)}$$

TotalCapacity is the total capacity of the coalition or the capacity of the agent that runs this algorithm

Graph G' is constructed this way, so that its edges are equal with $\frac{TimeSpent}{UtilityGained}$ and therefore, each edge represents how time efficient is to travel in that edge compared with other edges.

By running Dijkstra algorithm [4] on graph G' we can find the most efficient path to travel from a task to another task because utility over time, needed to complete a task and to travel to that task, is maximized and hence for each time unit spent, more utility is produced. Furthermore, by using the variable TotalCapacity, each time the algorithm runs, this variable has different value that reflects the capabilities of the coalition or the single agent that runs the DARRC algorithm. This way all the choices made by the algorithm take into consideration the current TotalCapacity and hence make more clever choices compared to other approaches.

However Dijkstra algorithm needs a minor modification in order to work in the RoboCup Rescue scenario. For each step(of the path) taken by the algorithm, the time needed to go to the next step is taken into consideration and only tasks that are possible to be completed before the deadline are considered as possible next steps of the algorithm. Also for each step of the algorithm it is calculated weather it is efficient to complete the task t of that step or not. This is checked by the following condition: if $(\frac{\frac{w(t)}{TotalCapacity}}{u(t)} < \frac{\frac{AverageWorkloadOfPath}{TotalCapacity} + \frac{AverageDistanceOfPath}{Speed}}{AverageUtilityOfPath}$ where n: the number of the tasks of the calculated path) then: complete the task t before moving to the next task. else using the original form of Dijkstra algorithm calculate the shortest distance from previous step of t to next step of t in the path, do not work on task t and travel directly to the next task of the path. The fraction: $\frac{\frac{w(t)}{TotalCapacity}}{u(t)}$ represents the expected time to complete the next task of the path over the amount of utility gained from this task. With this equation we can calculate how many time units are needed per each utility unit gained

The fraction: $\frac{\frac{AverageWorkloadOfPath}{TotalCapacity}+\frac{AverageDistanceOfPath}{Speed}}{AverageUtilityOfPath}$ represents the ratio of time needed to complete an average task plus the average time to travel to a task of the path over the average utility of the tasks of the path. This is fraction is used as a measure because we can calculate if it is better to complete the current task of the path and gain its utility or to travel to the next task and gain only the expected utility of the next task. Alternatively it represents the cost to jump to the next task without completing the task the current step. This equation is useful if a task, that belongs to the optimal path, needs a huge amount of time to be completed compared to its utility and with this equation we can determine if it is better to skip this task.

So the ModifiedDijkstra(G,i,j) algorithm takes as input a graph G and the nodes of G i and j. It returns a list named OptimalPath. Firstly, it produces the graph G' in the way that was described above and then the original Dijkstra algorithm runs in order to find the shortest path from node i to node j. After that, the original Dijkstra algorithm returns the shortest path, starting from node i and for each next node t of the path, the time needed to reach the next node of the path is calculated. For each step of the path, the condition $(\frac{\frac{w(t)}{TotalCapacity}}{u(t)} < \frac{\frac{AverageWorkloadOfPath}{TotalCapacity}+\frac{AverageDistanceOfPath}{Speed}}{AverageUtilityOfPath}$ where n: the number of the tasks of the calculated path) is checked. If this condition is true and task t can be completed before time, node t will be included in the list OptimalPath, else it will not be included in the list. If the previous condition is true and task t can not be completed before the deadline, ModifiedDijkstra will not consider the task t and re-run Dijkstra algorithm. This will continue to be repeated until we reach node j or if it is not possible to calculate any path with the remaining nodes. If the algorithm reached the node j, then it will return the list OptimalPath, else it will return -1.

6.3 Greedy Algorithm

The greedy algorithm that will be used in some cases of this algorithm, is the fast max sum algorithm [15] described earlier. The reason that this algorithm was chosen, is because it is considered the best distributed algorithm for the RoboCup Rescue problem based on simulations and the results it achieved over the other state of art algorithms.

6.4 Clustering Algorithm

The clustering algorithm that will be used is the algorithm described in a paper [6] proposed by Frey and Dueck. This algorithm is based on affinity propagation and it was chosen due to the fact that it can categorise graphs into clusters better and faster than any other available method.

6.5 Initialisation Algorithm

This is the algorithm that agents firstly run. Its main purpose is to compute the useful statistics defined above (section 6), find the complex tasks, initialise some variables and update the tasks list based on discovered tasks, completed tasks and the messages received by other agents. Furthermore in initialisation algorithm messages are sent to the other agents in order to inform them about new tasks. Finally and most importantly, the initialisation algorithm based on the existing situation decides which algorithm to run. If the agent is the leader of an existing coalition, Leader Algorithm is chosen. If the agent belongs in a coalition and is not the leader FollowerAlgorithm is chosen. If the agent was not chosen to be in a coalition, it will run the Not Chosen Algorithm. If the agent is not in a coalition and has the biggest priority then it is chosen as the leader of the coalition and will run the Start Leader Algorithm. Finally, If the agent that runs the algorithm does not have any communication with any other agent, it will run the No Communication Algorithm.

6.6 No Communication Algorithm

No communication algorithm runs whenever the agent that runs the proposed algorithm does not have any communication with any other agent. It runs the clustering algorithm in order to decide whether or not clusters, with sufficient UtilityOverTime exist. If a cluster with accepted $AverageUtilityOverTime$ is available and if the cost to reach such cluster is relatively low compared to the utility gained from this cluster, then the agent will follow the most efficient path to this cluster. Else the agent will run the greedy algorithm. In order for the agent to decide if it is efficient to visit a cluster the algorithm calculates some important parameters in lines 6-11. Greedy algorithm is used in order to calculate the total amount of utility that will be gained from tasks of the cluster before the deadline. After calculating the utility gained from the cluster, we can compare it to the total

Algorithm 1 Initialisation Algorithm

1. **compute** AverageWorkload
2. **compute**AverageCapacity
3. **compute** AverageDistance
4. **compute** AverageReachTime
5. **compute** AverageDeadline
6. **compute** AverageMaxWork
7. **compute** AverageTotalTasks
8. **compute** CapabilityFactor
9. **compute** AverageUtility
10. **compute** AverageUtilityOverTime
11. **compute** AverageWorkloadComplex
12. **run** find complex tasks
13. **set** $AssignedToComplexCapacity = 0$
14. **add** extra tasks discovered in the current list of tasks
15. **compare** received messages of other agents with the current list of tasks
16. **update** any outdated information about tasks
17. **send** information about tasks on current list of tasks to agents that have outdated information about any task
18. **if** $a_i \in C_j and C_j \in C$ {check if the agent belongs in a coalition} **then**
19. **if** $a_i \in \theta_l$ {check if the agent is the leader of the coalition} **then**
20. **run** Leader Algorithm
21. **else**
22. **run** Follower Algorithm
23. **end if**
24. **else**
25. **if** $a_i \notin \theta_n$ {check if the agent was not chosen anytime previously not to be in a coalition} **then**
26. **if** $\exists a_j$ and $comms(a_i, a_j) = 1$ {check if the agent can communicate with any other agent} **then**
27. **if** not $\exists a_j$ such that $j < i$ {check if the agent has the biggest priority} **then**
28. **run** Start Leader Algorithm
29. **else**
30. **run** Follower Algorithm
31. **end if**
32. **else**
33. **run** No Communication Algorithm
34. **end if**
35. **else**
36. **run** Not Chosen Algorithm
37. **end if**
38. **end if**

utility gained from all the path. The relationship between the TotalUtility and the ClusterUtility that is acceptable depends on the variable UtilityPer in line 11. If UtilityPer is equal with 0.8 it means that in order for a cluster to be accepted, the ClusterUtility must be equal or more than the 80 percent of the TotalUtility. UtilityPer is a variable that can be tuned depending on each scenario the algorithm runs, in order to perform better. Similarly ClusterLimit in line 3 can be tuned based on the type of the catastrophe, the topology of the map or the statistics of the city.

Algorithm 2 No Communication

1. **calculate** G' {In the way that was defined in the previous section}
2. **run** Clustering Algorithm
3. **if** ∃cluster and clusterAverageUtilityOverTime > Cluster_Limit {if the AverageUtilityOverTime of any cluster is considered satisfactory} **then**
4. j = the nearest task of the nearest cluster with clusterAverageUtilityOverTime > Cluster_Limit
5. **run** ModifiedDijkstraAlgorithm(G',CurrentLocation,j)
6. **calculate** PathUtility = total utility of the tasks that ModifiedDijkstra returned
7. **calculate** PathTime = total time needed to complete all tasks that ModifiedDijkstra returned
8. **calculate** Deadline2= Deadline - PathTime
9. **calculate** ClusterUtility = Utility gained from the cluster using Greedy Algorithm with deadline = Deadline2 and input tasks, the tasks that belong the nearest cluster only
10. **calculate** TotalUtility = PathUtility + ClusterUtility
11. **if** $ClusterUtility < TotalUtility \times UtilityPer$ {check if it is worth traveling to that cluster} **then**
12. **run** Greedy Algorithm
13. **else**
14. move to the nearest cluster using the shortest path calculated by Modified Dijkstra Algorithm (G',CurrentLocation,j)
15. **end if**
16. **else**
17. **run** Greedy Algorithm
18. **end if**

6.7 Not Chosen Algorithm

Not chosen algorithm runs whenever the agent that runs the proposed algorithm have communication with other agents but was not chosen to be member in any coalition. This algorithm follows the same steps with the No Communication Algorithm with a small difference. Due to the fact the the agent has the ability to

communicate with other agents, it takes into consideration other agents plans. If any other agent is expected to complete earlier any task that belongs to the path calculated by the agent that runs the algorithm, then the algorithm excludes these tasks and run the Not Chosen Algorithm again (line 15-21). After that, the agent broadcasts[B] the tasks of its calculated path with the expected completion time of each task(line 23). This way the agents will not attempt to complete tasks that other agents can accomplish faster and instead, can focus on tasks that no agent plans to visit and hence gain more utility globally.

6.8 Start Leader Algorithm

In the Initialisation Algorithm if an agent does not yet belong to a coalition and has the biggest priority is nominated as the leader. The StartLeaderAlgorithm based on the percentage of the workload of the complex tasks over not complex tasks and based on the workload of the agents that are available for communication, decides how many agents must be assigned in complex tasks (lines 1 and 2, variable y). Each time the algorithm runs it creates a coalition with total capacity equal to average workload of tasks plus standard deviation (line 5). This is because DARRC aims to create coalitions that can complete the majority of the complex tasks and for more difficult tasks, coalitions can cooperate and complete such tasks (future work). Depending on the capacity of agents already assigned to a coalition it decides whether or not another coalition will be created (lines 5 and 9). If an extra coalition will be created an agent from the remaining agents is nominated as leader and shall run Start leader algorithm later (line 12), else every agent left will not assigned to any coalition. In line 6 the agent with the biggest capacity, among the available agents, is selected because of a heuristic from [15] that suggests that it is better to send agents with more capabilities to tasks with more workload.

6.9 Follower Algorithm

It is the algorithm that each agent which is part of a coalition but not a leader will run. The only thing that this algorithm does is to follow orders from the leader of the coalition.

Algorithm 3 Not Chosen Algorithm

1. **calculate** G' {In the way that was defined in the previous section}
2. **run** Clustering Algorithm
3. **if** ∃cluster and clusterAverageUtilityOverTime > Cluster_Limit {if the AverageUtilityOverTime of any cluster is considered satisfactory} **then**
4. j = the nearest task of the nearest cluster with clusterAverageUtilityOverTime > Cluster_Limit
5. **run** ModifiedDijkstraAlgorithm(G',CurrentLocation,j)
6. **calculate** PathUtility = total utility of the tasks that ModifiedDijkstra returned
7. **calculate** PathTime = total time needed to complete all tasks that ModifiedDijkstra returned
8. **calculate** Deadline2= Deadline - PathTime
9. **calculate** ClusterUtility = Utility gained from the cluster using Greedy Algorithm with deadline = Deadline2 and input tasks, the tasks that belong the nearest cluster only
10. **calculate** TotalUtility = PathUtility + ClusterUtility
11. **if** $ClusterUtility < TotalUtility \times UtilityPer$ {check if it is worth traveling to that cluster} **then**
12. **run** Greedy Algorithm
13. **else**
14. calculate the most efficient path to the nearest cluster using the Modified Dijkstra Algorithm (G',CurrentLocation,j)
15. **for** every task t_i that belongs to the path **do**
16. **if** any of the other agents will complete t_i earlier than the agent that runs the algorithm **then**
17. exclude task t_i
18. **end if**
19. **end for**
20. **if** At least a task was excluded **then**
21. **run** Not Chosen Algorithm
22. **end if**
23. **broadcast** estimated time of completion of all tasks that belong to the calculated path and follow the path to the cluster
24. **end if**
25. **else**
26. **run** Greedy Algorithm
27. **end if**

Algorithm 4 Start Leader Algorithm

1. $x = \frac{TotalWorkloadComplex}{TotalWorkload} \times TotalCapacity$
2. $y = x - AssignedToComplexCapacity$
3. CapacityOfCoalition = 0
4. **if** TotalCapacity of avaialable agents > AverageWorkloadComplex+ Standart Deviation **and** y >AverageWorkloadComplex + Standart Deviation **then**
5. **while** CapacityOfCoalition <= AverageWorkloadComplex + Standard Deviation **do**
6. **select** the agent a_i with the biggest capacity between available agent and add a_i in the coalition
7. **inform** the agent a_i that it is a member of the coalition
8. AssignedToComplexCapacity = AssignedToComplexCapacity + $c(a_i)$
9. CapacityOfCoalition = CapacityOfCoalition + $c(a_i)$
10. **end while**
11. chosen agents will form a coalition with leader the agent that run the algorithm
12. **choose** agent with the biggest priority from the remaining agents inform it about the leadership of the next coalition and the variable AssignedToComplexTasks
13. **else**
14. All agents under the leader are chosen not to be in a coalition and coalition disband
15. **end if**

Algorithm 5 Follower Algorithm

1. **follow** leaders orders

6.10 Leader Algorithm

Its the algorithm that a leader of a coalition runs after all agents of a coalition are assigned. It is similar with NotChosenAlgorithm but it considers only complex tasks and communicates only with leaders of other coalitions.

7 Properties of the DARRC Algorithm

In this section some key properties of the DARRC algorithm are examined. Firstly it is proved that by running the modified Dijkstra algorithm on the graph G' the most efficient routes, between any two tasks, are calculated. After that it is proved that DARRC has equal performance with the Greedy algorithm in some specific cases. Next it is proved that DARRC can reach a cluster equally or more efficient compared to the Greedy algorithm in the rest of the cases. Finally some discussion is being made regarding on how the value of some variables can change

Algorithm 6 Leader Algorithm

1. **consider** only complex tasks
2. **calculate** G' {In the way that was defined in the previous section}
3. **run** Clustering Algorithm
4. **if** ∃cluster and clusterAverageUtilityOverTime > Cluster_Limit {if the AverageUtilityOverTime of any cluster is considered satisfactory} **then**
5. j = the nearest task of the nearest cluster with clusterAverageUtilityOverTime > Cluster_Limit
6. **run** ModifiedDijkstraAlgorithm(G',CurrentLocation,j)
7. **calculate** PathUtility = total utility of the tasks that ModifiedDijkstra returned
8. **calculate** PathTime = total time needed to complete all tasks that ModifiedDijkstra returned
9. **calculate** Deadline2= Deadline - PathTime
10. **calculate** ClusterUtility = Utility gained from the cluster using Greedy Algorithm with deadline = Deadline2 and input tasks, the tasks that belong the nearest cluster only
11. **calculate** TotalUtility = PathUtility + ClusterUtility
12. **if** $ClusterUtility < TotalUtility \times UtilityPer$ {check if it is worth traveling to that cluster} **then**
13. **run** Greedy Algorithm
14. **else**
15. calculate the most efficient path to the nearest cluster using the Modified Dijkstra Algorithm (G',CurrentLocation,j)
16. **for** every task t_i that belongs to the path **do**
17. **if** any of the other agents will complete t_i earlier than the agent that runs the algorithm **then**
18. exclude task t_i
19. **end if**
20. **end for**
21. **if** At least a task was excluded **then**
22. **run** Leader Algorithm
23. **end if**
24. **broadcast** estimated time of completion of all tasks that belong to the calculated path and follow the path to the cluster
25. **end if**
26. **else**
27. **run** Greedy Algorithm
28. **end if**
29. Send message to all of the other members of the coalition with the directions of the next task on the path that was calculated in previous steps

the performance of the proposed algorithm and how these variables can be tuned in order to outperform the Greedy algorithm

Lemma 7.1 (Efficiency in Traveling). The algorithm guaranties the maximum possible utility over time gained to travel from a place to another

Proof. 1. Given the fact that the weights of the graph G' are calculated based the formula: $ij = \frac{\frac{w(j)}{TotalCapacity} + \frac{distance(ij)}{speed}}{u(j)}$, where i and j are tasks

Because of this formula, the weights of the graph G' represent the total time spent over utility gained. This is true because the $TotalTimeSpent = CompletionTimeOfTask + TimeToTravelToTask$.$CompeltionTimeOfTask = \frac{w(j)}{TotalCapacity}$ and $TimeToTravelToTask = \frac{distance(ij)}{speed}$.

2. Dijkstra algorithm guarantees that the calculated path between 2 locations of a graph has the minimum possible weight of all the possible paths between the two locations.

3. Given the facts 1 and 2 along with the 2 modifications on Dijkstra algorithm described earlier, it is guaranteed that the maximum possible utility over time gained is gained using this algorithm. □

Lemma 7.2 (Equal Performance with Greedy Algorithm Case No Communication). In case of no communication with any other agent and if any of the following conditions is true:

1. No cluster with $AverageUtilityOverTime > ClusterLimit$ exists.

2. The Utility of the cluster is less than the expected TotalUtility gained from the path to, and the cluster multiplied by the UtilityPer variable.

The algorithm guaranties equal performance with the greedy algorithm.

Proof. If the agent is unable to communicate with any other agent then No Communication Algorithm is selected to run. If condition 1 is true, No Communication Algorithm will select the greedy algorithm to run and hence the results of the two algorithms will be exactly the same. If condition 2 is true, No Communication Algorithm will select the greedy algorithm to run and hence the results of the two algorithms will be exactly the same.

□

Lemma 7.3 (Equal Performance with Greedy Algorithm Case Not Selected). In case that an agent is not selected to be in a coalition and if any of the following conditions is true:

1. No cluster with $AverageUtilityOverTime > ClusterLimit$ exists.

2. The Utility of the cluster is less than the expected TotalUtility gained from the path to, and the cluster multiplied by the UtilityPer variable.

The algorithm guaranties equal performance with the greedy algorithm.

Proof. If the agent was not selected to be in a coalition then Not Chosen Algorithm is selected to run. If condition 1 is true, Not Chosen Algorithm will select the greedy algorithm to run and hence the results of the two algorithms will be exactly the same. If condition 2 is true, Not Chosen Algorithm will select the greedy algorithm to run and hence the results of the two algorithms will be exactly the same.

□

Lemma 7.4 (Equal Performance with Greedy Algorithm Coalition Leader). In case that an agent is a leader of a coalition and if any of the following conditions is true (for the complex tasks):

1. No cluster with $AverageUtilityOverTime > ClusterLimit$ exists.

2. The Utility of the cluster is less than the expected TotalUtility gained from the path to, and the cluster multiplied by the UtilityPer variable.

The algorithm guaranties equal performance with the greedy algorithm.

Proof. If the agent was not selected to be in a coalition then Leader Algorithm is selected to run. If condition 1 is true, Leader Algorithm will select the greedy algorithm to run and hence the results of the two algorithms will be exactly the same. If condition 2 is true, Leader Algorithm will select the greedy algorithm to run and hence the results of the two algorithms will be exactly the same.

□

Lemma 7.5 (Reach closest cluster more or equally efficient compared with the greedy algorithm. Case No Communication). In case of no communication and there exists at least one cluster with $AverageUtilityOverTime > ClusterLimit$

and the Utility of this cluster is more than $TotalUtility \times UtilityPer$. The algorithm guaranties to reach a cluster equally or more efficient than the greedy algorithm.

Proof. In case of no communication then No Communication Algorithm is selected to run. If there exists at least one cluster with $AverageUtilityOverTime > ClusterLimit$ and the Utility of this cluster is more than $TotalUtility \times UtilityPer$, DARRC will calculate the path to the closest cluster using the modified Dijkstra Algorithm. Based on Lemma 1, it was proved that the path that was calculated by the modified Dijkstra Algorithm is the most efficient from all possible paths. Because of the previous statement it is proved that in the best case scenario for Greedy Algorithm, it will choose a path with the same efficiency with the DARRC algorithm. However the chance of doing this is:

$$ChanceOfSameChoice = \frac{1}{AverageDegreeOfGraph}^{|StepsOfPath|}$$

This is a very small chance and therefore, most of the times DARRC is expected to travel to a cluster more efficiently than the Greedy Algorithm or in the worst case scenario, equally efficient with the Greedy Algorithm. The rest of the cases DARRC will reach the closest cluster more efficiently compared to the greedy algorithm.

□

Lemma 7.6 (Reach closest cluster more or equally efficient compared with the greedy algorithm. Case Not Selected in Coalition). In case that an agent was not selected in a coalition and there exists at least one cluster with $AverageUtilityOverTime > ClusterLimit$ and the Utility of this cluster is more than $TotalUtility \times UtilityPer$. The algorithm guaranties to reach a cluster equally or more efficient than the greedy algorithm.

Proof. In case that an agent was not selected in a coalition then Not Chosen Algorithm is selected to run. If there exists at least one cluster with $AverageUtilityOverTime > ClusterLimit$ and the Utility of this cluster is more than $TotalUtility \times UtilityPer$, DARRC will calculate the path to the closest cluster using the modified Dijkstra Algorithm. Based on Lemma 1, it was proved that the path that was calculated by the modified Dijkstra Algorithm is the most efficient from all possible paths. Because of the previous statement it is proved that in the best case scenario for Greedy Algorithm, it will choose a path with the same efficiency with the DARRC algorithm. However the chance of doing this is:

$$ChanceOfSameChoice = \frac{1}{AverageDegreeOfGraph}^{|StepsOfPath|}$$

This is a very small chance and therefore, most of the times DARRC is expected to travel to a cluster more efficiently than the Greedy Algorithm or in the worst case scenario, equally efficient with the Greedy Algorithm. The rest of the cases DARRC will reach the closest cluster more efficiently compared to the greedy algorithm.

□

Lemma 7.7 (Reach closest cluster more or equally efficient compared with the greedy algorithm. Case Leader of a Coalition). In case that an agent is the leader of a coalition and there exists at least one cluster of complex tasks with $AverageUtilityOverTime > ClusterLimit$ and the Utility of this cluster is more than $TotalUtility \times UtilityPer$. The algorithm guaranties to reach a cluster equally or more efficient than the greedy algorithm.

Proof. In case that an agent was not selected in a coalition then Leader Algorithm is selected to run. If there exists at least one cluster with $AverageUtilityOverTime > ClusterLimit$ and the Utility of this cluster is more than $TotalUtility \times UtilityPer$, DARRC will calculate the path to the closest cluster using the modified Dijkstra Algorithm. Based on Lemma 1, it was proved that the path that was calculated by the modified Dijkstra Algorithm is the most efficient from all possible paths. Because of the previous statement it is proved that in the best case scenario for Greedy Algorithm, it will choose a path with the same efficiency with the DARRC algorithm. However the chance of doing this is:

$$ChanceOfSameChoice = \frac{1}{AverageDegreeOfGraph}^{|StepsOfPath|}$$

This is a very small chance and therefore, most of the times DARRC is expected to travel to a cluster more efficiently than the Greedy Algorithm or in the worst case scenario, equally efficient with the Greedy Algorithm. The rest of the cases DARRC will reach the closest cluster more efficiently compared to the greedy algorithm.

□

Lemma 7.8. The DARRC algorithm will reach the closest cluster equally or more efficient compared to the greedy algorithm in any possible scenario.

Proof. From lemma 5,6 and 7 it is proved that the DARRC algorithm will reach the closest cluster equally or more efficient compared to the greedy algorithm in

the cases of no communication, not chosen to be in a coalition and leader of the coalition. These 3 cases are all the possible cases and hence for any possible case DARRC will perform equal or better than the greedy algorithm. □

Theorem 7.9. The DARRC algorithm will perform better on average, compared with the greedy algorithm.

Proof. Lemma 8 proves that the proposed algorithm will reach the closest cluster equally or more efficient compared to the greedy algorithm in any possible scenario. However the performance of the algorithm depends in the type of the destruction and how some variables are set. By setting some variables over a specific number it is guaranteed that the DARRC algorithm will outperform greedy algorithm on average. The efficiency on travelling inside a cluster Depends on how close $ClusterLimit$ is compared to global $AverageUtilityOverTime$. If $ClusterLimit$ is set to be higher than $AverageUtilityOverTime + StandartDeviation$, it will be much more efficient to complete tasks that belong to the cluster and move to that cluster, rather than move based on the greedy algorithms decisions. The Greedy algorithm is expected to choose a path with $UtilityOverTime$ slightly more than $AverageUtilityOverTime$. If a high $ClusterLimit$ is combined with a high value of the $UtilityPer$ variable, for example 0.9 then this means that the 90% of the total utility gained from the path will be gained inside a cluster. Because of this the probability of performing better than the Greedy Algorithm in this case is increased. By increasing the values of both of the $ClusterLimit$ and $UtilityPer$, it increases the chance to perform better than the Greedy Algorithm. However if any of these two variables is set in a very high value, the performance of the proposed algorithm will be equal with Greedy Algorithm, because no Cluster will match the criteria of the proposed algorithm and hence the Greedy Algorithm will be selected to run. In this case the proposed algorithm will perform exactly the same with the Greedy Algorithm as it was proved in Lemma 2,3 and 4.

□

8 Evaluation

As it was proved in Lemma 1, DARRC can calculate the most efficient path between any two tasks. This is a major contribution in the area of RoboCup Rescue, because the same procedure to build the alternative graph and the modified version of Dijkstra can be used by future approaches that want to move agents from

one location to another. By using this method it is proved and hence guaranteed that the path between the two locations is the most efficient path.

Furthermore, by using graph G' and the modified version of Dijkstra algorithm, it is guaranteed and proved by lemmas 5,6 and 7 that DARRC will reach a cluster equally or more efficient compared to the Greedy algorithm, which is currently the best distributed algorithm available.

Many people may argue that it is not important to reach a cluster. However, this is not true for all the cases and it depends on the location of the tasks and the topology of the map. In some cities there exist some main areas, that during certain hours (e.g. between 8 a.m. and 4 p.m.) each day, a large amount of people are gathered there compared with other locations. These could include places where people work, shop or can be entertained. For example in Southampton such locations could be Portswood Street and High Street. If a catastrophe happens during rush hours then each person will be a task that must be completed by the rescue force. Because civilians are gathered in popular places then each popular place would become a cluster. Also, clusters of tasks could be formed by a disaster, for example, in a war after bombing, after a large scale terrorists attack or many pieces of a meteoroid fall in an area.

Depending on how some variables are tuned in DARRC algorithm, an agent can follow the path to a cluster or run the Greedy algorithm as was discussed in lemma 8. This, together with the big number of civilians gathered in the same places gives a big advantage to DARRC over the Greedy algorithm because clusters will exist and DARRC will reach them more efficiently. After that, when the agent will be inside a cluster, it will gain more utility compared to other non cluster tasks. Additionally in the case that clusters do not exist, DARRC will run the Greedy algorithm, and hence have exactly the same performance.

All things considered, DARRC in certain cases will have equally or better performance compared to Greedy algorithm and in the remainder cases will have equal performance, therefore DARRC will be more efficient overall.

9 Future Work

Sometimes, during a rescue operation in real life, after saving a civilian it may be necessary to transfer him in a hospital. However, none of the existing approaches have dealt with this issue. This can be expressed in existing models by increasing

the workload of a task but it would be better to plan which tasks to complete and also consider the locations of the hospitals, in order to save time by bringing the patient faster to a hospital or give higher priority to tasks near a hospital for this reason.

Some tasks may be very critical and rescue force has to complete such tasks because after their deadline they may cause more damage and create more tasks. In real world this could be a nuclear energy factory, a big fire that can spread to other buildings and some buildings may contain explosives and if they get fire more damage may be caused. A good algorithm should be able to do what is more efficient considering these facts.

Up to date, many methods that deal with the RoboCup rescue problem were proposed. Most of them perform differently based on the topology of the tasks, the number of tasks, the density and other parameters. Instead of trying to create one approach that will be more efficient than any other approach for all the possible cases, which is impossible, it would be better to use a hybrid approach. For example, Greedy algorithm could be used when the environment is highly dynamic, since Greedy algorithm does not plan many steps forward in contrast with the method that uses Markov models or other statistic methods. This will waste precious time to plan future optimal plans but the environment may change very fast and the plan may no longer be optimal. Also, the opposite can happen in a non dynamic environment. For this reason, it is better to use different methods depending on each occasion. This can also be done for the communication range, clusters of tasks etc. By using multiple methods will automatically boost the performance of a proposed algorithm. This is the reason that DARRC sometimes runs the F-Max-Sum algorithm.

Moreover, future work can focus on methods that use statistics and probabilities to plan their next moves. An agent could calculate the probability of new tasks per location. For example, if many tasks are in the same place it is possible that more tasks will be discovered in the future in the same place or nearby that place. Probabilities could also be useful to calculate the next moves of adjacent agents and move in a way that global utility will be maximised (e.g., do not go to a task that will be completed soon by another agent). Also for each city that the algorithm is intended to be used, a list of what is included in each building and how many people are there on average may be available. By using statistics and probabilities and based on the type of the inside of a building (explosives, flammable, etc) deadline and utility can be calculated. Moreover, the timing

of the disaster can sometimes be important in the topology of the tasks. For example, if something happens during peak hours the majority of civilians will be in major roads. In addition, sometimes information about new tasks received by an agent may be inaccurate due to communication problems or human error. The probability for any new information that was received could be calculated based on data on previous tasks or previous disasters.

Furthermore, most of the papers in this area are evaluated using a simulator. However, the simulator may have bugs or cover only a small number of possible cases. This way the proposed algorithms could be tuned in order to perform well in a simulator, but in real cases the proposed methods may not be efficient at all. This can be avoided by testing the proposed algorithms on different simulators. Also no existing simulator uses data from a real disaster. Statistics from a catastrophe of the past could be used in order to create a more reliable simulator and hence achieve better results. Another possible solution is to divide the map into smaller areas and assign each area to an agent or a coalition depending on the workload of the task on each area.

Neural Networks could be used for tuning some important variables of DARRC based on previous data. For example, ClusterLimit and UtilityPer variables could be the output of a neural network that has input the characteristics of the current tasks and was trained using data from real life destructions or simulations.

DARRC in the future could be extended in order to support coordination between different type of agents and with tasks having different deadlines. Also, leaders of a coalition could negotiate with other leaders by sending messages defining how much extra utility would be gained if their coalition had extra x amount of capacity. Then, each leader could calculate whether it would be better for global utility to send agents to another coalition, or to wait for help from other coalitions. Also, two or more coalitions could cooperate in order to complete more difficult tasks. This could create a second level of coalition.

10 Conclusion

The main purpose of this thesis was the development of an algorithm for the RoboCup Rescue problem. At the beginning, the state of the art approaches in the area of RoboCup Rescue were investigated together with the pros and cons of the most important Centralised, Decentralised and Coalition approaches.

Furthermore, a new model was proposed based on the model described in [15]. Moreover, a novel algorithm, DARRC that plans its moves based on clusters of tasks was proposed. In addition, it was proved that DARRC can calculate the most efficient, based on the percentage of time needed over utility gained, path between any two tasks. This is a major contribution in the area of the RoboCup Rescue, because in the future any proposed algorithm that needs a method, to calculate a path from a task to another task, can use the DARRC algorithm. Also, it was proved that agents that run the DARRC algorithm will reach a cluster of tasks equally or more efficiently compared to the Greedy algorithm. Nevertheless, it was examined how some critical variables that are used by DARRC could be tuned in order to achieve better overall performance compared to the Greedy algorithm. Finally, certain ideas that could be used in the future to improve DARRC or help the improvement of the performance , were discussed.

Personal Reflection of Thesis

The learning outcomes derived from this project included the history of the RoboCup Rescue Project and the investigation of the state of art methods. In addition, I learned from where these methods were derived (based on other methods) and the cases that each approach is better compared to similar approaches. It was also important to understand the differences between the assumptions that each algorithm considers, and what is the impact of these assumptions on the design and the performance of the proposed solutions. Another key outcome was to learn how to critically read papers about algorithms and evaluate each method, taking into consideration the assumptions that have been made. Furthermore, I read about some brilliant ideas that can be developed further and form solutions that can actually be used in real world destructions and save much more lives, compared with solutions that rescue forces use up to date. Additionally I learned how to use latex, Bibtex and Lyx, in order to produce well organised and better looking papers.

It should be noted that it was difficult for me to become familiar with Latex so I preferred to learn how to use the Lyx text editor. The Lyx was supposed to be able to do almost everything that the Latex could do. However, some things that were critical for this thesis were impossible to be completed by using Lyx (i.e., writing algorithms).

Additionally, the available time (three months) was not enough, considering the

fact that a topic should be chosen, literature search should be completed and a model, different from other models, should be created. Adding to this, I should have though about different ideas for the chosen model, choose the best one, develop it, improve it and finally evaluate it. This is the main reason why the evaluation of the proposed algorithm was performed using theoretical proofs instead of empirically evaluate it, by using a simulator. In order to provide an empirical evaluation, the proposed method should be implemented, the clustering algorithm should be implemented, the algorithms that the method was going to be compared with, should be implemented and finally I should have learnt how to use an existing simulator for the RoboCup Rescue problem.

Appendix A

Alternative Methods

Methods that are described in this section were also designed for this thesis, but instead this project was focused in the DARRC algorithm because there was not enough time available. DARRC method was chosen because it is more promising compared with the other approaches and because it was possible to theoretically prove some properties of the DARRC algorithm.

Priority Based on Deadline Urgency

In this method, tasks can have only a predefined number of values as deadlines. The difference with the DARRC algorithm is that the deadline is not fixed for all the tasks. For example, tasks could be divided in the following groups: Urgent tasks (tasks with deadline less than x), Important tasks (tasks with deadline less than $2 \times x$ and more or equal with x) and NonImportant tasks (tasks that have deadline more or equal with $2 \times x$) where x is a predefined amount of time units. Based on this assumption and based on the $CapabilityFactor$, the algorithm can decide if it has enough time to complete all the tasks. For example, if the $CapabilityFactor = 1.2$ then it means that the algorithm has time to complete 120% of the tasks. If the algorithm has enough time it will consider only the tasks that have urgent deadline until the deadline of the urgent tasks is over and only then, it will consider the remaining tasks. This way if the algorithm has enough time for Important and NonImportant tasks, it can give priority to urgent tasks and complete them before their deadline expires hence maximising the total utility gained. This method is more realistic compared with other methods that can have deadlines with any possible value. This is due to the fact that in real

world it is impossible to decide with big detail how many time units a deadline must be equal to. In contrast it is easier and more reliable to distinguish deadline of tasks in a limited number of categories and define criteria for each category.

Appendix B

Messages Format

In this section the most important forms of messages and their fields, that are used in the DARRC algorithm, are described.

Discovery of Tasks and Planning

This section focus in describing the fields of the messages that agents sent in the Initialisation Algorithm, in order to inform other agents about new tasks, completed tasks and generally any other change. Also the same messages format is used from Not Chosen Algorithm and Leader Algorithm in order to inform other agents about the expected completion time of tasks are planned to be visited.

- Agent/Coalition number - A number that uniquely identifies agents and coalitions.

- Task Coordinates - Two numbers that denote the location (x,y) of a task.

- Task Utility - A number that represents how much Utility will be gained after completing each task before the deadline.

- Task Workload - A number that represents how much work a task needs in order to be completed.

- Task Deadline - A number that represents the maximum amount of time units that a task must be completed within, in order to get its Utility.

- Estimated Completion Time - A number that represents the time estimated for a task to be completed. Will be a negative number if this task is not in the plan of the agent or coalition. Will return 0 if this task is already completed or its deadline ended.

These are the fields that each message for Tasks Discovery and Planning shall contain.

Negotiation between Coalitions

This section focus in describing the messages that will be send from Future approaches, in order for coalition leaders to be able to negotiate about transferring agents to or from other coalitions.

- Coalition Number - A number that uniquely identifies a coalition.

- Extra Capacity Needed - A number that represents the amount of extra capacity that a coalition needs, in order to complete one extra task before deadline.

- Extra Utility Gained From Help - A number that represents the extra Utility that this coalition will gain if it will receive extra capacity equal or more with the amount denoted in the previous field.

- Time Left - A number that represents the amount of time units that help must be received within, in order to be useful.

These are the fields that each message for Negotiation between Coalitions shall contain.

Appendix C

Time Plan and Management

Week # *week beginning :*	1 13/6	2 20/6	3 27/6	4 4/7	5 11/7	6 18/7	7 25/7	8 1/8	9 8/8	10 15/8	11 22/8	12 29/8	13 5/9	14 12/9	15 19/9
Activities and milestones															
Decide Topic of Thesis															
Background Research and Literature review															
Analyse Existing Solutions															
Create Representation															
Create Solution															
Evaluation															
Writing-up															
Milestone – demonstrate to sup/examiner														*	
Milestone – dissertation draft complete														*	
Final corrections															
Milestone – Hand-in															*

Bibliography

[1] S. Balakirsky. Usarsim: Providing a framework for multi-robot performance evaluation. In *In: Proceedings of PerMIS*, pages 98–102, 2006.

[2] Stephen Balakirsky, Stefano Carpin, and Arnoud Visser. Evaluating the robocup 2009 virtual robot rescue competition. In *Proceedings of the 9th Workshop on Performance Metrics for Intelligent Systems*, PerMIS '09, pages 109–114, New York, NY, USA, 2009. ACM.

[3] Archie C. Chapman, Rosa Anna Micillo, Ramachandra Kota, and Nicholas R. Jennings. Decentralised dynamic task allocation: a practical game: theoretic approach. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 915–922, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[4] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 10.1007/BF01386390.

[5] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, pages 639–646, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.

[6] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[7] Erol Gelenbe and Stelios Timotheou. Random neural networks with synchronized interactions. *Neural Comput.*, 20:2308–2324, September 2008.

[8] Hiroaki Kitano. Robocup rescue: A grand challenge for multi-agent systems. *Multi-Agent Systems, International Conference on*, 0:0005, 2000.

[9] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, AGENTS '97, pages 340–347, New York, NY, USA, 1997. ACM.

[10] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. Robocup: A challenge problem for ai. *AI Magazine*, 18, 1997.

[11] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawai, and Hitoshi Matsubara. Robocup: A challenge problem for ai and robotics. In *RoboCup-97: Robot Soccer World Cup I*, Lecture Notes in Computer Science. 1998.

[12] Max Pfingsthorn, Bayu Slamet, and Arnoud Visser. A scalable hybrid multi-robot slam method for highly detailed maps. In *RoboCup 2007: Robot Soccer World Cup XI*. Springer Berlin / Heidelberg, 2008.

[13] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Trans. Comput.*, 38:1110–1123, August 1989.

[14] S. D. Ramchurn, M. Allen-williams, I. Vetsikas, A. Rogers, R. K. Dash, P. Dutta, and N. R. Jennings. Aladdin-rescue team description.

[15] Sarvapali D. Ramchurn, Ro Farinelli, Kathryn S, Maria Polukarov, and Nicholas R. Jennings. Decentralised coordination in robocup rescue, 2009.

[16] Sarvapali D. Ramchurn, Maria Polukarov, Alessandro Farinelli, Cuong Truong, and Nicholas R. Jennings. Coalition formation with spatial and temporal constraints. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 3 - Volume 3*, AAMAS '10, pages 1181–1188, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[17] Wei Ren and Nathan Sorensen. Distributed coordination architecture for multi-robot formation control. *Robotics and Autonomous Systems*, 56(4):324 – 333, 2008.

[18] Martijn Rooker and Andreas Birk. Combining exploration and ad-hoc networking in robocup rescue. In *RoboCup 2004: Robot Soccer World Cup VIII*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005.

[19] Xiaoming Zheng and Sven Koenig. Reaction functions for task allocation to cooperative agents. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, pages 559–566, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.