# RMASBench: Benchmarking Dynamic Multi-Agent Coordination in Urban Search and Rescue

Paper ID: #208

## ABSTRACT

We propose RMASBench, a new benchmarking tool based on the RoboCup Rescue Agent simulation system, to easily compare coordination approaches in controlled settings for dynamic rescue scenario. In particular, we offer simple interfaces to plug-in coordination algorithms without the need for implementing and tuning low-level agents' behaviors. Moreover, we add to the realism of the simulation by providing a large scale crowd simulator, which exploits GPUs parallel architecture, to simulate the behaviour of thousands of agents in real time. Hence, we present two key benchmarks for coordination algorithms. First we focus on a specific coordination problem where fire fighters must combat fires and prevent them from spreading across the city. We formalize this problem as a Distributed Constraint Optimization Problem and we compare two state-of-the art solution techniques: DSA and Max-Sum and thus provide the first benchmarks for DCOPs in this domain. Second, we provide key results on the evacuation of crowds of 3000 civilians on different maps under different blockage conditions. Thus we define benchmark scenarios for the development of coordination algorithms for large scale crowd evacuation. Our results demonstrate that RMASBench offers powerful tools to compare coordination algorithms in a dynamic environment.

## 1. INTRODUCTION

The development of benchmarking platforms for agent-based systems is a fundamental step towards building agent-based applications in the real world. Benchmarking platforms allow us to evaluate the performance of state-of-the-art algorithms and mechanisms in controlled settings and determine the best ones to use under a wide range of realistic conditions. To this end, these platforms need to incorporate simulation environments that pose realistic challenges that mimic those of the real world. This means creating environments where the problems are dynamic, the environment variables may be liable to bias or uncertainty and where, potentially, agents have to address issues of scale. Examples of such platforms include multi-agent coordination competition platforms [1, 20] and the TAC platforms [7].[1]

Crucially, such platforms need to allow for easy implementation of new algorithms without requiring researchers to implement low-level elements irrelevant to their algorithm. To date, very few benchmarking platforms have been developed according to such principles (and adopted by the artificial intelligence community).[2] In particular, we note the dearth of realistic testbeds for agent-based coordination mechamisms such as distributed constraints optimisation (DCOP), task allocation (TA), or coalition formation (CF). Those testbeds that do aim to evaluate such algorithms (e.g., DCOPolis[3], CATS[4]) typically provide inputs that are drawn from fixed distributions or define coordination problems that are usually static or require significant extensions to create dynamic settings and large-scale problems. As a result, there are currently no well defined realistic benchmarks for DCOP, TA, and CF.

Against this background, we develop and evaluate a novel testbed for multi-agent coordination algorithms called RMASBench. Our work builds upon the existing RoboCup Rescue simulation platform (RSP) that simulates an urban search and rescue scenario and has also been used by emergency responders and planners to both train and plan for emergencies [19]. In the RSP, agent designers have to code police agents to unblock roads, fire brigade agents to extinguish fires, and ambulance agents to rescue trapped civilians. We choose such a platform because it creates a realistic simulation environment that presents significant aspects of dynamism (e.g., fires spread across a city, crowds of evacuees move according to their changing priorities), uncertainty (e.g., the behaviour of fires and civilians is determined by a number of factors that may not be perfectly sensed or modeled), and issues of scale (e.g., hundreds or thousands of civilians may need to be saved, or directed to refuges while hundreds of fires may need to be contained) [8]. More importantly, the problems that the RSP requires agent designers to solve for, include, but are not limited to, coalition formation (e.g., forming teams of ambulances or fire brigades to save civilians and extinguish fires respectively) [15], distributed constraint optimisation (e.g., to ensure police agents unblock roads in an optimal way and ambulances and fire brigade agents work together to maximise lives saved) [2], and task allocation (e.g., to schedule allocation of tasks according to levels of priority) in general. However, the RSP as it is, does not allow for easy implementation of coordination algorithms, often requiring coding low-level elements such as network communication protocols and path-planning algorithms. Moreover, the heavy computations involved in generating realistic simulations (e.g., large crowds moving or fires spreading) require multiple machines and only permit simulations with low numbers of agents (fewer than 200).

[1]Altogether these competitions attract hundreds of participants every year and contribute to the development of solutions to real-world problems and the advancement of research in general.

**Appears in:** *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.

[2]This is partly due to the significant effort required to develop such platforms. For example, testbeds such as RoboCupRescue, TAC, TAC-SCM or PowerTAC have involved large long-term investments to be completed and require large teams to code for.

[3]`http://www.dcopolis.org/`

[4]`http://www.cs.ubc.ca/~kevinlb/CATS/`

Hence, as a foundational step to address these issues and turn the RSP into an integrated testbed, we significantly extended the RSP in order to allow for the easy implementation of DCOP algorithms and develop a large-scale crowd simulator that allows users to run simulations involving thousands of agents in real-time on a single standard computer. In more detail, our work advances in the following ways. First, we formalize the fire brigade coordination problem as a DCOP and develop a test suite for DCOP algorithms. In particular, our test suite provides a simple communication protocol for the agents and provides facilities to compute standard metrics for evaluating DCOP approaches. Second, we implemented standard DCOP algorithms such as DSA [6] and Max-Sum [5] on our test suite, and show how they perform under different simulation settings. In more detail, we show that a standard factor graph representation of our scenario requires a prohibitive amount of computation for the MaxSum agents. Hence, we propose a number of heuristics to prune the factor graph and show that this results in a significant computation reduction and good performance. Third, we develop a large-scale crowd simulator based on the use of Graphics Processing Units (GPUs) that allows us to scale the simulation to thousands of agents. Fourth, we present estimates of crowd evacuations for different maps for 3000 civilians given different conditions of road blockage. In so doing, we establish benchmark scenarios, and thus the initial steps towards building a complete testbed for multi-agent coordination algorithms.

The remainder of this paper is organised as follows. In Section 2 the benchmark is described. The fire fighting coordination problem and the coordination mechanisms we used are presented in Section 3 and in Section 4 we provide a description of the crowd simulator component. In Section 5 results from experiments are presented and we finally conclude with Section 6.

## 2. RESCUE AGENT COMPETITION AND RMASBENCH

Over the last few years, several techniques for multi-agent strategy planning and team coordination has been proposed [13, 14, 9] and there has been substantial work on building information infrastructure and decision support systems for enabling incident commanders to efficiently coordinate rescue teams in the field. For example, Schurr *et al.* introduced a system based on software developed in the RSP competitions for the training and support of incident commanders in Los Angeles [19].

Recently, the RMASBench was developed as part of the RSP to introduce a generic API for multi-agent coordination that essentially provides facilities for exchanging messages among agents and for making coordinated decisions. Moreover, RMASBench provides a library implementing state-of-the art solvers for DCOPs such as *DSA* and *MaxSum*, as well as facilities for comparing coordination algorithms. Both solvers and their integration in RMASBench are described in Section 3.1.

To implement new coordination algorithms, users can simply derive derive a class from the *DecentralizedAssignment* interface, which essentially represents an agent, and implement the following functions. i) **computeOutgoingMessages()**; ii) **getIncomingMessages()**; iii) **computeAssignment()**.

Note that each of these functions will then be called in a synchronized manner within each coordination cycle of the assignment computation. In more details, Figure 1 depicts the embedding of RMASBench into the agent simulation package of the RSL. Since communication in the existing RSP is limited to a few messages within a simulated hour of the scenario, the *ComDispatcher* has been introduced to extend the existing communication mechanism of the simulation kernel. In fact, state-of-the-art coordination approaches (e.g., DCOP solvers) typically require much more message interactions per simulation step. The *ComDispatcher* can be seen as an additional communication layer that connects agents outside the RSP. As shown in Figure 1, the dispatcher is executed on top of the *Center Agent* which acts as a proxy to the simulator. In this way, also centralized coordination schemes are possible by implementing a *CentralizedAssignment* interface on the *Center Agent* directly.

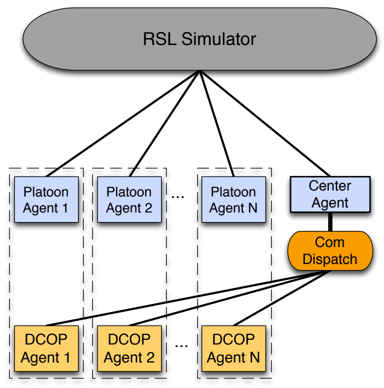

Figure 1: Functional connection between the rescue simulator (grey & blue) and RMASBench (orange).

After either centralized or decentralized decision making via message exchange has been concluded, results are communicated by the *Center Agent* (the centres for police, fire, and ambulance agents) to the *Platoon Agents* (i.e., police, fire brigade, and ambulance), which then adjust their actions accordingly. While the *ComDispatcher* allows messages at higher bandwidth and frequency than the standard communication mechanism of the RSP, it also facilitates the measurement of frequency and bandwidth usage over time, and their comparison for the various coordination approaches within the framework.

Against this background, in the next section we present a framework to evaluate DCOP algorithms for task allocation among fire brigade agents. To this end, we first present both a formal model of the problem and algorithms to solve it. By so doing, we aim to establish key benchmarks for DCOPs in dynamic environments.

## 3. FIRE BRIGADE COORDINATION

In this section we focus on a specific problem instance of RMASBench and consider the coordination problem of assigning fire brigades to fires so to mitigate damages to buildings. We focus on this problem because it poses crucial challenges for coordination: i) it is a dynamic problem: the set of fires to be extinguished varies over time because fires may spread to neighbouring buildings and grow in intensity as time proceeds ii) it has a strong spatial aspect: a fire brigade agent can operate on a given fire only if it is physically located in the proximity of that fire, hence agents must consider their travel time when coordinating and iii) it includes coalitional effects: several agents can work on the same fire, and the more agents work on the same fire the faster the fire can be extinguished. However if too many units are allocated to the same fire they might hinder each other or ignore other fires.

Similar features are present also in the allocation of groups of ambulances to rescue civilians [14]. However with respect to this scenario, the fire brigade coordination problem is harder to decompose. In fact, while in the ambulance scenario the deadline for rescuing a civilian clearly defines which agents can be considered for the allocation (i.e., the ones that can reach the civilian's location before the deadline), in most fire scenarios all fire brigades can be useful to extinguish all fires in the city. This is because fires spread to neighbouring building and hence even if a rescue agent can not reach a fire location before the building is completely burned down, it still makes sense for the fire brigade to be allocated to the fire as it could be of help in extinguishing fires that spread to neighbouring buildings.

Following a common practice in the literature [13, 18, 4] we cast this problem as an iterative task assignment problem, where fire fighters are allocated to fires by solving a series of task assignment problems that are assumed to be independent one from the other. Such task assignment problem has been solved by using various techniques such as auctions [13], token passing [18] and biologically-inspired approaches [4].

Here, we focus on DCOP solution techniques for two main reasons: i) the DCOP framework allows to efficiently solve complex problems by exploiting domain structure; ii) there are a rich wealth of solution techniques for DCOPs (both exact and approximate) that are often compared and evaluated on synthetic benchmarking scenarios (e.g., graph colouring). We believe that in this sense RMASBench is an ideal testing ground to compare such techniques considering various performance metrics (e.g., solution quality, communication and computation overhead) and different operative conditions.

## 3.1 The Fire Brigade Coordination Problem

We indicate with $FA$ the set of fire brigade agents that must be allocated to a set of fires $F$. We indicate with $u_j(\mathbf{a})$ the *utility* achieved by the system when the subset $\mathbf{a} \in FA$ is allocated to fire $f_j \in F$, such utility is a performance estimation for the subset of agents allocated to the fire. Given the above discussion, to optimally solve this task assignment problem one must be able to estimate the performance of groups of agents for each fire, and then solve a coalition formation problem based on this estimation. However, coalition formation in its most general form is known to be a hard optimization problem [15], therefore, here we make a number of assumptions to provide a tractable formulation of this problem. First, we assume that fire brigade agents are able to estimate the maximum number of fire units that should be allocated to each fire $f_j$ ($maxAg_j$). Second, if the group size is less than this maximum then we assume that the group utility is the sum of the utility of each single unit, i.e., $u_j(\mathbf{a}) = \sum_{a_i \in \mathbf{a}} u_{i,j} - M\phi(\mathbf{a})$. Where, $u_{i,j}$ is the utility of allocating the fire unit $a_i$ to fire $f_j$, $\phi(\mathbf{a})$ is an indicator function that equals one when $|\mathbf{a}| > maxAg_j$ and $M$ is a big number. Both $u_{i,j}$ and $maxAg_j$ depend on several domain specific parameters, however, since the focus of this work is on benchmarking of coordination approaches rather than finding better strategies for the RSP scenario, here we adopt standard heuristics used in the competitions. In particular, in our empirical analysis, the utility of allocating a fire unit $a_i$ to a fire $f_j$ is based on the intensity of the fire (i.e., the fieriness computed by the RSP), and the travel costs needed by agent $a_i$ to reach fire $f_i$. The quantity $maxAg_j$ also depends on the size of the building and the on the amount of area that can be extinguished by a single agent.

Considering the mentioned assumption, this coordination problem can be conveniently represented by using a DCOP formulation. A standard DCOP formulation comprises a set of agents $\mathcal{A}$ that control a set of variables $\mathcal{X}$. Typically each agent can control a set of variables, but each variable is controlled by only one agent. Variables range over a set of discrete and finite domains $\mathcal{D}$, and each variable $x_i$ can take values in its corresponding domain $D_i$. Finally, $\mathcal{C}$ is a set of functions that represent constraints among variables. Each function $C_j : D_{j_1} \times \cdots \times D_{j_{r_j}} \rightarrow \Re \cup \{-\infty\}$ assigns a real value for each possible joint assignment of the variables it depends on ($\mathbf{x}_i \subseteq \mathcal{X}$). Given this, agents aim at finding the joint variable assignment that maximizes the sum of constraint functions, i.e. $\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{C_j \in \mathcal{C}} C_j(\mathbf{x}_j)$. To solve this maximization task, agents can communicate by exchanging messages, however, each agent $a_i$ can communicate with an agent $a_j$ only if $x_i$ and $x_j$ belong to the scope of at least one constraint function. The subset of agents that can communicate with $a_i$ are the *neighbours* of $a_i$. Following the approach in [14], here we represent each fire brigade unit with one variable that encodes the current task the agent is assigned to. Variable domains are fires that can be assigned to the agent, and we have one constraint function for each fire that represents the above defined utility for that fire. With a slight abuse of notation we denote such utility as $u_j(\mathbf{x}_j)$, where $\mathbf{x}_j \subseteq \mathcal{X}$ represent the variables of the subset of agents that could be allocated to fire $f_j$.

## 3.2 Coordination Mechanisms

As mentioned before, there are many techniques that can be applied to solve DCOPs. Such techniques, range from exact algorithms [12] that are guaranteed to provide the optimal solution, to low cost sub-optimal approaches [6, 5]. Here we focus on the latter to meet the real time constraint of our scenario. In particular we implement and evaluate two state-of-the-art approaches: DSA [6] and MaxSum [5]. We choose these algorithms because they represent two widely used approximate techniques that propose different approaches for solving DCOPs: DSA is essentially a greedy local search, while MaxSum is based on inference. In what follows, we briefly detail the two approaches.

### *The DSA algorithm.*

DSA is essentially a greedy local search executed in parallel by all the agents. In DSA, parallel execution is controlled by introducing a stochastic decision on whether performing a local assignment change. In more detail, for each execution step of the RoboCup Rescue simulator there is an initialization and several DSA coordination cycles: in the initialization phase each agent randomly chooses a value for its variable and then communicate this value to all its neighbours, then, at each coordination cycle each agent executes the following operations: i) receive messages from neighbours and update information accordingly; ii) choose an activation probability $ap_i \in [0, 1]$; iii) choose a random number $rn_i \in [0, 1]$ iv) if $rn_i < ap_i$, choose a value for the variable that optimizes the local gain and if the value differs from previous assignment send a message to all neighbours notifying the change. DSA executes this coordination cycle until convergence (i.e., until there are no changes in the joint assignment) or for a predefined number of iterations (in our experiments on average the number of coordination cycles to converge is about 48 and the maximum number of iterations are 500).

Following a common practice in the DCOP community [6], here we use a single activation probability for all the agents and we fix this once for all the mission execution. Moreover, we operate in a synchronous execution model, where all agents execute their action in parallel and then communicate.

A distinctive feature of DSA is the extremely low overhead both in terms of communication and computation. In fact, each DSA agent sends at most one message to all its neighbours per coordination cycle and such message only contains the current variable assignment. As for computation, each agent maximizes the local gain only with respect to possible assignments of its variable, hence the maximization procedure that each agent executes must only perform $|D_i|$ function evaluations. However, DSA's performance in dynamic settings such as the one we consider here is highly dependent on the choice of the activation probability, and even with a fine tuning of such parameter the algorithm suffers from local maxima.

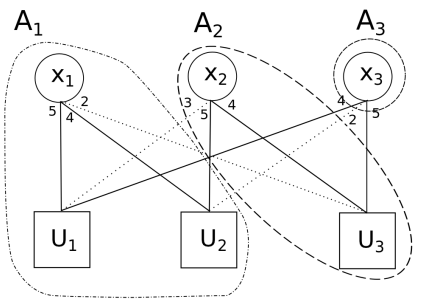

Figure 2: A fully connected factor graph with three variables and three functions. Numbers on the edges represent the utility of allocating the agent (corresponding to the variable node) to the fire (corresponding to the function node); Dashed lines represent links removed by the factor graph reducing procedure (with $k = m = 2$). Ovals represent the allocation of variable and function nodes to agents.

## *The MaxSum algorithm.*

The MaxSum algorithm [5] offers a very different perspective for decentralized optimization as it is essentially based on inference: each agent tries to compute an estimation of the impact that each of its action has on the global optimization function (usually called the $z_i(x_i)$ function), by exchanging messages with its neighbours. Once the $z_i(x_i)$ function is built each agent chooses the variable value that maximizes this function. The operations of the MaxSum algorithm are best understood on a factor graph representation of the agents' interactions. A factor graph is a bipartite graph with two types of nodes: variable and function nodes, where each function node is linked to a variable node iff the variable belongs to the function's scope (Figure 2 shows an exemplar factor graph with three fires and three fire brigades). To compute the mentioned $z_i(x_i)$ function, the MaxSum algorithm exchanges two kinds of messages: variable to function messages ($q_{i\to j}(x_i)$) and function to variable messages ($r_{j\to i}(x_i)$). We report here the equation for the MaxSum message update and refer the reader to [3] for a pseudo-code description of the algorithm:

$$r_{j\to i}(x_i) = \max_{\mathbf{x}_{j,-i}}(u_j(x_i, \mathbf{x}_{j,-i}) + \sum_{k\in\mathcal{N}_j\setminus i} q_{k\to j}(x_k)) \quad (1)$$

$$q_{i\to j}(x_i) = \alpha_{ij} + \sum_{k\in\mathcal{M}_i\setminus j} r_{k\to i}(x_i) \quad (2)$$

here $\mathcal{N}_j$ ($\mathcal{M}_i$) represent the set of variable (function) node indices that are connected to function node $j$ (variable node $i$) and $\alpha_{ij}$ is a scaler chosen such that $\sum_{x_i} q_{i\to j}(x_i) = 0$[5]. Finally, $z_i(x_i) = \sum_{k\in\mathcal{M}_i} r_{k\to i}(x_i)$. The MaxSum algorithm requires the agents to perform the computation related to message update for variable and function nodes. The allocation of variable nodes to agents is straightforward, because one variable is always controlled by only one agent. On the other hand, agents must decide who is in charge for the computation related to shared function nodes. However, notice that while such allocation impacts on load distribution across the agents, it has no effect on the solution quality and on communication overhead. Therefore, following [14] here we allocate computation for shared function to the agent with the lowest ID among the ones that are involved in that constraint function (see Figure 2 for an example). Next, each agent initializes all messages to zero and, at each time step, it computes new messages based on last received messages. When this iterative process terminates each agent computes the $z_i(x_i)$ function and assigns to $x_i$ the value that maximizes this function. Such iterative process terminates when the messages reach a fixed point (i.e., new messages are identical to previous messages). However, the MaxSum algorithm is guaranteed to converge to a fixed point only if the factor graph is acyclic and in this case it computes the optimal assignment. In more general settings, there are only limited guarantees on convergence and solution quality, however extensive empirical results [5] show that when executed on a loopy factor graph MaxSum often achieves very good solutions. Since in these cases convergence is not guaranteed the iteration process is usually performed for an arbitrary (relatively small) number of coordination cycles (30 iterations in our experiments).[6] Notice that, the message update for a function to variable message $r_{j\to i}(x_i)$ (Equation 1) involves a maximization that ranges over all the possible joint assignments of $\mathbf{x}_j$. Therefore, a straightforward implementation would cycle through $d^{|\mathbf{x}_j|}$ possible assignments where $d$ is the size of the variables' domain.

[5]we refer to [5] for further details on these equations.

## *MaxSum refinements for dynamic problem settings.*

If we directly model our domain with a factor graph as discussed above, we would have that each function should be connected with all the agents in the system. This is because for most of the rescue scenarios, every agent can potentially reach every fire in the city and then work on it. Hence, given the above analysis, the computational effort associated to MaxSum operations would become prohibitive, especially considering the real time requirements associated with our scenario.

Given the complexity in computing the optimal allocation of agents to fires in terms of maximising the number of fires to extinguish, in minimum time, and using the most useful agents (closer and with resources), we employ a number of heuristics that address each of these dimensions and show how these result in a significant reduction in computation without significant loss in quality of solutions. First, in order to maximise the number of fires extinguished, we restrict the degree of factors by $k$ in order to ensure a fair spread of agents across fires. Second, to maximise the number of agents per fire, we in turn bound the degree of each variable by $m$. Third, to ensure that the most useful agents are selected for each fire, we restrict the degree of each variable by considering the factors that represent fires with highest single agent utility ($u_{i,j}$) for the agent owning those variables. By so doing, we essentially balance the factor graph and, in turn, bound the computation associated with function to variable message update by $m^k$ (because $|\mathbf{x}_j| \leq k$ and $d \leq m$).

In more details, we consider as possible targets for an agent $a_i$ only a subset of $k$ fires that belongs to $F$. Such $k$ fires are computed for each agent $a_i$, by ordering the set of fires in decreasing order of single agent utility $u_{i,j}$ and by considering only the top $m$ elements. The same criterion is used to select variables that belong to the scope of each constraint function and selecting the top $k$. The result is a factor graph where each variable is linked to at most $m$ function nodes and each function node is linked to at most $k$ variable nodes. [7] Notice that, if the number of fire brigades significantly outnumbers the fires some fire brigades might remain idle (i.e., some variable nodes might not be linked to any function). If this happens we send these fire brigade agents to their best target

[6]In the experiments we performed the MaxSum joint assignment converged roughly 70% of the time.

[7]We note that the factor graph reduction does require a distributed ad-hoc procedure that each agent runs before executing MaxSum . In the interest of space, we do not report here this procedure, but we do consider this preprocessing phase when evaluating the coordination overhead of MaxSum in the empirical section.

(i.e., they do not coordinate).

To further reduce the computational effort related to message update, we use a particular function encoding, suggested by the approach taken in [14], which exploits the specific structure of our constraint functions. In more detail, in our reference domain, the value of a function changes significantly only for specific assignments of the variables and remains unchanged for several others. In particular, consider the function $u_1(x_1, x_2, x_3)$ in Figure 2. The value of $u_1(f_1, f_1, f_3)$ is clearly different from the value of $u_1(f_1, f_2, f_3)$, but $u_1(f_1, f_2, f_3) = u_1(f_1, f_3, f_3)$. In other words, for $u_j(.)$ all the allocations in which a variable is not allocated to $f_j$ but to *any other fire* are equivalent. We can exploit this insight by restricting the domain of variables that participate in a constraint function $u_j(.)$ to only two values, which indicate whether the variable is assigned to fire $f_j$ or not. This has a significant impact on the total number of tuples that must be considered by the optimization procedure as we go from a number of $k^{|\mathbf{x}_j|}$ to $2^{|\mathbf{x}_j|}$.

## 4. LARGE SCALE EVACUATION SIMULATION

Having described the framework within which coordination algorithms can be easily implemented into the RMASbench, we now turn to the issue of adding more realism to the types of problems that agents need to solve for. In particular, we are interested in creating scenarios where agents have to cope with issues of scale by having to coordinate the movement of a large number of evacuees in the disaster rescue setting. This task is performed by police agents in the RSP and is much more complex than the fire fighter coordination (as presented earlier) given the highly distributed, dynamic, and complex behaviours of crowds. Thus, while crowd evacuation represents a challenge for coordination mechanisms, it goes well beyond the scope of this paper.[8] Instead, here we focus on how to create such complex scenarios to implement realistic simulations and provide initial intuitions as to how such a problem might be modelled and solved by the coordination algorithms presented earlier.

Given that the RSP only allows sequential computation, i.e. simulating agents and civilians' behaviour sequentially by processing every agent's commands (e.g., path planning, messaging etc..) on a single CPU, it can take a significant amount of time to simulate small numbers of agents moving on very small maps. Where the RSP might be used by emergency planners to formulate evacuation plans, it means that there are only a limited number of simulations that can be run if a limited amount of time is availble to emergency planners in realistic settings. Moreover, for researchers wishing to evaluate their coordination algorithms on the RSP, such evaluations can only be very limited as well.

To address these issues and equip both practitioners and researchers with an efficient benchmarking platform, we implemented an evacuation simulator that implement individual civilians as individual agents running their commands in parallel. Crucially, we implemented agents' path planning and social behaviours in terms of a parallel program that can be run on standard Graphics Processing Units for easy deployment on any PC [17]. By so doing, we are able to exploit the highly parallelised memory and processing architecture of the GPU to speed up the simulations by orders of magnitude and therefore create realistic situations for decentralised algorithms to solve in real time. This is in line with similar recent simulation environments are based on GPU Programming (using the CUDA language provided by NVIDIA), such as [10]. In what follows, we detail the key elements of the approach.

[8]We leave the extension of distributed coordination formalisms for crowd evacuation to future work.

### *Loading Maps and Agent Data.*

A key step in performing computations on the GPU involves loading the data required by the algorithms onto the graphcis card. In the case of evacuation simulation, we need to import the map data and civilians' data into the graphics memory (which is typically limited to 1GB).[9] To this end, we transform the map data (typically provided in GML – Geographic Markup Language – format) into a valid format for the GPU. This data includes buildings, roads, and the networks over them. After importing the map data into graphics card memory, we store the information about civilians, which include the initial positions, body sizes, speeds and so on.

### *Parallel Implementation of Evacuation Behaviours.*

The problem of simulating realistic evacuation behaviours on a large scale requires computing evacuation paths (to refuges) for every single agent (civilian) in the simulation and, as the agents move along these paths, adapt their behaviour with those they happen to meet on the way. This may take into account the simulated velocities as well as the social interactions that may exist among them. We discuss each of these in turn.

To compute the path of each agent, we implement the routing algorithm of civilians in the disaster environment, by which each civilian can find the path from a specific building or road to the refuges.[10] Specifically, we use a Breadth-First-Search algorithm to compute the path from each building or road to the refuges, and then save these paths into the graphics card global memory which civilian can access during the simulation.

Now, to simulate the actual movement of each civilian according to other forces around it, from each possible building or road to the refuges, we introduce physical forces model that guides the movement of civilians according to the combination of a number of attractive and repulsive forces that exist around each of them.

### *Attraction to Refuges.*

This force dictates the natural movement one would expect for all civilians in the enviroment in a disaster scenario and is the strongest force. However, civilians cannot move towards the refuges directly but can only move to a position which is in their line of sight. Therefore, we need to determine the next target point for a civilian (building or road) based on its current speed, its maximum speed and the paths to refuges. After determining the next target point, we can determine the acceleration given by the attraction force to refugee, as follows (replace $x$ with $y$ to obtain $AD_y$):

$$AD_x = (V_{max} \times \frac{dx}{\sqrt{dx^2 + dy^2}} - V_x) \times \delta \qquad (3)$$

where $V_{max}$ is the max velocity that the civilian can reach, $V_x$ and $V_y$ are the civilian's velocity at the $x$-axis and $y$-axis respectively, $dx, dy$ are the distances from the starting point to the centroid of the next target point respectively, and $\delta$ is an acceleration coefficient where $\frac{1}{\delta}$ means the time (in ms) required by the civilian to reach its maximum velocity. Now this acceleration interacts with other forces as follows.

### *Forces due to interaction with other civilians.*

[9]As a first step, we load the data into the *global* memory of the card as opposed to specific block registers. While the global memory is not the fastest in terms of data access for a GPU, it permits individual threads to share data seamlessly.

[10]These paths can be pre-calculated before the start of the simulation to save time.

This includes civilians pushing each other or avoiding collision. To this end, We assume that the interaction of civilians follows the Boids *flocking model* [16]. Specifically, the flocking model includes three simple rules: (1) Cohesion: steering towards the average position of neighbours (2) Alignment: steering towards the average heading of neighbours; (3) Separation: avoiding crowding neighbours. A civilian's behaviour is only affected by civilians within a specific neighbourhood or locus around it, which is deemed its neighbour. Typically, in flocking models, rules 1) and 2) adopt the same neighbourhood, and rule 3) adopts a smaller neighbourhood to exhibit grouping behaviours.

We now describe these three rules in detail. For a specific civilian $a$, we assume that its neighbourhood used in rule 1) nd 2) is denoted as a set $C_1$, and the neighbourhood adopted in rule 3) is denoted as $C_2$. Now for the civilians in $C_1$, we compute the average interaction centre's $x$ and $y$ coordinates (replace $x$ with $y$ to get the y-coordinate) $NC_x = \frac{\sum_{c \in C_1} c.x}{|C_1|}$, where $c.x$ is $x$ coordinate for civilian $c$. Given this, we can compute the average velocity of the neighbours $C_1$ at $x$ and $y$-axis respectively, (replace $x$ with $y$ to obtain $NV_y$) as $NV_x = \frac{\sum_{c \in C_1} v_c.x}{|C_1|}$, where $v_c.x$ and $v_c.y$ are the velocity at $x$ and $y$ axis for civilian $c$.

The average collision centre can then be calculated as (replace $x$ with $y$ to obtain $CO_y$) $CO_x = \frac{\sum_{c \in C_2} c.x}{|C_2|}$

From the average centre, average velocity and average collision centres, we can compute the acceleration for civilian $a$ caused by the three rules. The acceleration in Euclidian space generated by the rule 1) is (replace $x$ with $y$ to obtain $ANC_y$):

$$ANC_x = (V_{max} \times \frac{NC_x - a.x}{\sqrt{(NC_x - a.x)^2 + (NC_y - a.y)^2}} - V_x) \times \delta$$

The acceleration generated by the rule 2) is (replace $x$ with $y$ to obtain $ANV_y$):

$$ANV_x = (V_{max} \times \frac{NV_x \times T}{\sqrt{(NC_x \times T)^2 + (NC_y \times T)^2}} - V_x) \times \delta$$

where $T$ is the amount of time taken by a time step in the simulation. Finally the acceleration generated by rule 3) is (replace $x$ with $y$ to obtain $ACO_y$):

$$ACO_x = (V_{max} \times \frac{a.x - CO_x}{\sqrt{(a.x - CO_x)^2 + (a.y - CO_y)^2}} - V_x) \times \delta$$

Note that in the above equation $a.x - CO_x$ indicates that the civilian moves away from the collision centre.

Now, we can compute the overall acceleration generated by the flocking model as:

$$FL_x = ANC_x + ANV_x + ACO_x$$
$$FL_y = ANC_y + ANV_y + ACO_y$$

### *Forces due to Physical Objects.*

Finally, we compute the repulsive force from walls. Given the civilian's acceleration towards the next target point and the acceleration due to the flocking model, we can compute the expected velocity of the civilian at $x$ and $y$-axis respectively:

$$EV_x = V_x + (AD_x + FL_x) \times TEV_y = V_y + (AD_y + FL_y) \times T \tag{4}$$

Then according to this velocity, we check whether the civilian moves towards a wall. If this is the case, and the distance is less than the civilian's body size, the wall applies a force to stop the civilian. Otherwise, the civilian moves for distance $EV_x \times T$ and $EV_y \times T$ on the $x$ and $y$-axis.

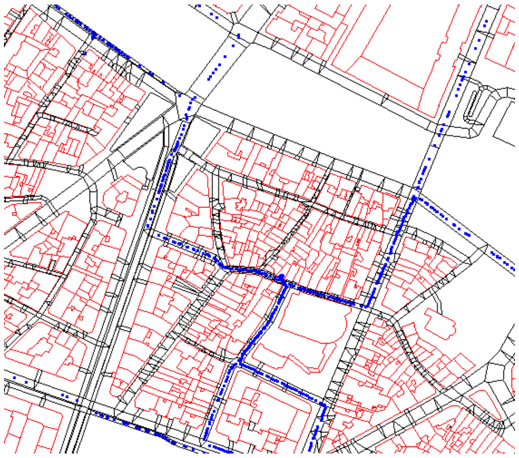
Figure 3: Part of the map of Paris with 2000 civilians simulated in real-time on a GPU.

Given the above model (and as we show later in Section 5.2), by running the above parallel algorithms on a GPU, we are able to simulate thousands of civilians moving in *faster than real time* – where, for example one second step in the simulator equals more than two seconds worth of movement by the civilians. This is considerably faster and more memory efficient (requiring no more than a a few MBs to hold a map and civilian data) than the previous RSP simulator which was also limited to 200 agents on a normal PC.

Another main advantage of running the evacuation simulator on the GPU is that the data it generates can be directly rendered to the user interface. Thus, to visualise the map and the civilians' movements we use CUDA libraries and OpenGL.

### *Integration with RMASBench.*

Finally, the evacuation simulator was integrated with RMASBench in such a way that both platforms can be run on different PCs to allow for efficient parallel computations at the platform level (espousing the distributed architecture adopted by the RSP). This was achieved by using a socket connection to transmit data between the two. The communication protocol between RMASBench and the crowd simulator is based on XML and allows to exchange information about the simulated agents' positions at every time step. In more detail, RMASBench sends a first message with the initial positions of all the civilians and rescue agents in the map. At each subsequent simulation step, the RMASBench sends a request for the next position of each agent and using this information, it then evolves the simulation updating the status of all agents, including their health status (which dictates if they can move or are trapped), and messages they can hear (from other agents close by).

In the Section 5.2, we show how our simulator allows for large scale evacuation simulations in real-time and provide some initial insights into how different maps may influence the evacuation process.

## 5. EMPIRICAL RESULTS

Having described the main components of RMASBench, we now present our empirical analysis for the fire brigade coordination problem (5.1) and for the civilian evacuation scenario (5.2).

## 5.1 Fire Fighting Coordination

We now report empirical results for the comparison of MaxSum and DSA in the fire fighting coordination problem described in Section 3.1. In particular, for MaxSum we use the factor graph restriction described in Section 3.2 and we set $k = m$ to simplify the test setting. For DSA, we assume no restriction on fire reachability for the fire brigades, i.e., any agent can work on any fire present in the city. This essentially results in a fully connected constraint graph where every agent is a direct neighbour of every other agent. Finally, we also consider another version of DSA where the neighbouring relationship is computed on the restricted factor graph, we call this method DSA-R. In DSA-R two agents are neighbours if they can work on the same fire according to the restricted factor graph representation described in 3.2[11]. We empirically tuned the activation probability of DSA and in all the following experiments we use the value 0.7 that achieved the best results in our testing scenario (in the interest of space figures related to this tuning are not reported here). To evaluate algorithms' performance we consider standard metrics for the RoboCup scenario. In particular, given our focus on fire fighting operations we measured and analysed the following metrics: i) the number of destroyed buildings, ii) the square meters of damaged areas, iii) the number of building that are burning at each point in time, iv) the number of buildings that caught fire at least once during the simulation (*onceburned* ) and v) the extinguish time, i.e. the time step at which all fires have been extinguished. We report here data about the *onceburned* and the extinguish time metrics because they summarize very well the performance of our coordination mechanisms. Moreover, we consider, as a performance measure for coordination, the number of times we allocate a number of agents different to $MaxAg_j$ for each fire $f_j$ (number of violated constraints).

As for coordination overhead we use standard measures for evaluating DCOPs, such as the number of messages exchanged per simulation step, the size of messages exchanged (in bytes) per simulation step and the Non Concurrent Constraint Checks (NCCCs), which is a standard measure for computation overhead in DCOPs [11]. We focused on two reference scenarios: i) the Kobe map with 3 ignition points and 12 fire brigade units and ii) the Paris map with 3 ignition points and 21 fire brigades units. Each simulation is run for 300 steps and the start time for fire brigades is after 40 time steps in Kobe and 23 in Paris. Before that time fire brigades remain idle and after that time they become aware of all fires' locations in the city map.

To evaluate the impact of the factor graph restriction, we performed a set of experiments were we vary the parameter $k$. Simulation results reported in Figure 4 show that when $k$ increases the performance increases (i.e., *onceburned* metric decreases), however, the difference between DSA-R and MaxSum for $k = 3$ and $k = 4$ are significantly smaller. Moreover, results show that by using our proposed heuristics for factor graph pruning the algorithms achieve good performance with relatively low values of $k$. Based on these results we will use $k = 4$ in the following experiments.

Table 1 and Table 2 report a comparison of the three coordination algorithms based on the main metrics we discussed above for Kobe and Paris respectively. Results show that MaxSum outperforms both DSA and DSA-R for the *onceburned* metric, extinguish time and violated constraints. This is particularly evident in the Paris scenario. In Paris good allocations in the first time steps of the simulation are crucial, because, if fires start spreading it quickly becomes impossible to contain them. This can be seen by the fact that the standard error of the mean for MaxSum in Paris is much higher relatively to the absolute value of the *onceburned* metric. This higher value is explained by many runs with good *onceburned* values and few with very poor performance. Hence, overall results suggest that DSA and DSA-R make poor decisions in the first crucial steps and consequently fail to contain the fires. The results related to coordination overhead (message number, message size and NCCC) confirm that MaxSum requires significantly more computation and communication than both versions of DSA (especially for what concerns the NCCCs). However, comparing the DSA and DSA-R we can see that by using the factor graph pruning heuristics we can achieve similar performance while significantly reducing coordination overhead (e.g., we reduce the size of total message exchanged by more than one order of magnitude). This suggests that our factor graph pruning approach is not limited to MaxSum but could be successfully applied to other DCOP solvers.

[11]Note that, for DSA-R the factor graph is only used to compute the neighbouring relations. We could use any other approach to choose the neighbours, and we opted for this one to be coherent with the MaxSum algorithm and have a fair comparison.

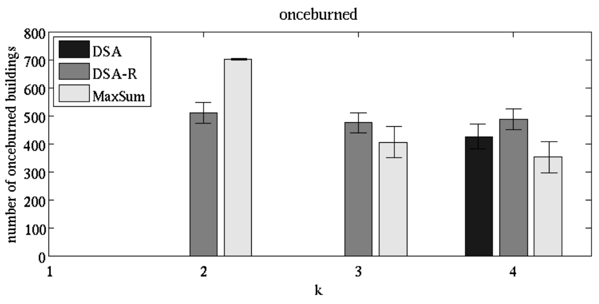

Figure 4: Comparison of the *onceburned* metric for DSA, DSA-R and MaxSum plotted against the parameter $k$. Results are averaged over 30 runs. Error bars report the standard error of the mean.

| Metrics/Algs | DSA | DSA-R | MaxSum |
|---|---|---|---|
| Ext. Time | 261,6 [$\pm$11.06] | 276.7 [$\pm$7.56] | **230.73** [$\pm$15.11] |
| *onceburned* | 425,6 [$\pm$44.69] | 486.7 [$\pm$38.09] | **352.2** [$\pm$55.76] |
| Violated Const. | 35,83 [$\pm$0.98] | 35.5 [$\pm$0.85] | **17.3** [$\pm$0.42] |
| Msgs num. | 5547,85 [$\pm$47.16] | **119.3** [$\pm$5.47] | 2950.4 [$\pm$35.74] |
| Msgs bytes | 44382 [$\pm$377.23] | **954.66** [$\pm$43.75] | 65683 [$\pm$807.47] |
| NCCC | **122,27** [$\pm$8.64] | 166.3 [$\pm$8.96] | 166782 [$\pm$4179.74] |

Table 1: Statistics for DSA, DSA-R and MaxSum averaged over 30 runs for the Kobe scenario. For DSA-R and MaxSum $k = 4$. The best result for each metric is in bold. Numbers in brackets report the standard error of the mean

## 5.2 Simulation Evaluation

In this section, we demonstrate that our crowd simulator can create realistic dynamic behaviours for large scale simulations. The scale factor is particularly important to address scenarios that can be useful for rescue operators in real applications.In more detail, we simulated the evacuation of 3000 agents 3000 on maps of Berlin and Paris. An example of the runs on the Paris map is given in Figure 3 showing congestion at one of the major arteries of the city.

To evaluate how different unblocking strategies might perform in this scenario, rather than implement a specific police strategy, we create artificial roadblocks across the maps to create settings where police agents may be successful at unblocking roads to different degrees. This is because our focus is on the simulation properties rather than the unblocking strategies. Thus, we varied the

| Metrics/Algs | DSA | DSA-R | MaxSum |
|---|---|---|---|
| Exti. Time | 273.85 [$\pm$8.63] | 299 [$\pm$0] | **134.6** [$\pm$13.12] |
| *onceburned* | 1132.95 [$\pm$65.42] | 1110.15 [$\pm$81.57] | **168.25** [$\pm$81.47] |
| Violated Const. | 89.62 [$\pm$2.26] | 126.41 [$\pm$2.72] | **46.2** [$\pm$1.03] |
| Msgs num. | 8853.25 [$\pm$147.43] | **143.18** [$\pm$5.28] | 3767.4 [$\pm$153, 33] |
| Msgs bytes | 70826 [$\pm$1179.5] | **1145.47** [$\pm$42.26] | 99880 [$\pm$1995.2] |
| NCCC | **240.28** [$\pm$23.49] | 341.19 [$\pm$30.52] | 263608 [$\pm$6218.4] |

Table 2: Statistics for DSA, DSA-R and MaxSum averaged over 20 runs in the Paris scenario. For DSA-R and MaxSum $k = 4$. The best result for each metric is in bold. Numbers in brackets report the standard error of the mean

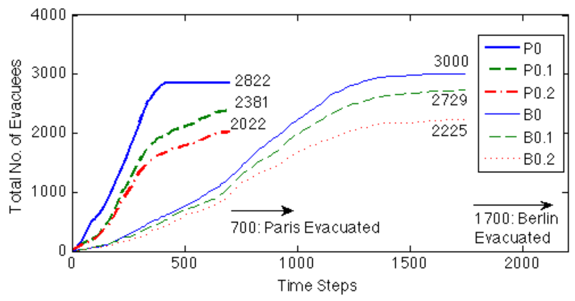

Figure 5: Simulation results from runs on Berlin and Paris maps with 3000 civilians. B$k$ and P$k$ represent Berlin and Paris for different blockage ratios $k$.

ratio of the number of blocked roads (to the total number of roads between 0 to 0.2 with step size 0.1 (which we call the blocked ratio). Going beyond 0.2 would result in the map being occasionally disconnected (by blockages) and therefore impossible to evacuate. For each blocked ratio and the total number of blocked roads, we compute the actual number of blocked roads, and then randomly choose this number of the roads and set them as blocked. Furthermore, we assume that maximum velocity of agents is uniformly distributed between $1m/s$ to $3m/s$ and each time step corresponds to $1s$ (i.e., the simulator is queried for positions at every $1s$ of simulation time). Then, for each map and each certain called blocked ratio, we run the simulation for 10 times. We record the mean total evacuation time. The simulation results are shown in Figure 5. As can be seen, the evacuation times for the Berlin map are nearly three times (1700 time steps compared to 400 for 0 blockage ratio) those of the Paris map. Moreover, a high percentage of civilians (nearly 30% in Paris) tend to completely trapped in parts of the city (with Paris being worse than Berlin) and never manage to evacuate given the blocked roads. A closer inspection at the maps reveals that the Berlin map has many more junctions than Paris and, while this causes regular congestions (leading to long evacuation times), it also allows many more civilians to escape since it has more alternatives at every junction. Moreover, we note that Paris has very low evacuation times (around 400) when there are no blockages while Berlin has a high evacuation time no matter what blocked ratio applies. From the Paris map (see Figure 3), it can be seen that there are a number of key bridges that, if blocked, will cause major congestions on other bridges dividing the city in two. The effect of randomly blocking even one of these was found to double the evacuation time (see the table above). Thus, if all bridges are blocked, the evacuation never completes as agents are trapped on either side of the city.

These results point to the complexity that agents need to deal with when coordinating to unblock key roads across a given city. In particular, we note that such coordination would require modelling the impact of the roads that connect otherwise disconnected parts of the city, the distance travelled by individuals across large numbers of junctions, and the starting positions of those civilians (which we randomly simulated in our case). Moreover, the agents would have to model and predict the movement of thousands of civilians in real time if an optimal solution is to be achieved. Hence, be believe our simulator poses interesting challenges for the development of efficient coordination algorithms.

## 6. CONCLUSIONS AND FUTURE WORK

We introduced RMASBench, a testbed for multi-agent coordination based on the RoboCup Rescue Simulation Platform. The new testbed provides extremely challenging problems for multi-agent coordination in the USAR domain, and, in contrast to existing testbeds, it offers highly dynamic and large-scale benchmarking environments. Using RMASBench we demonstrated the evaluation of state-of-the-art algorithms for the fire fighting problem and we simulated large scale evacuation scenarios on realistic maps, setting key benchmarks in this domain.

Future work in this space includes the integration of other coordination techniques and the development and evaluation of coordinated approaches for crowd evacuations.

## 7. REFERENCES

[1] T. Behrens, M. Köster, F. Schlesinger, J. Dix, and J. Hübner. The multi-agent programming contest 2011: A résumé. In L. Dennis, O. Boissier, and R. Bordini, editors, *Programming Multi-Agent Systems*, volume 7217 of *Lecture Notes in Computer Science*, pages 155–172. Springer Berlin / Heidelberg, 2012.

[2] A. Chapman, R. A. Micillo, R. Kota, and N. Jennings. Decentralised dynamic task allocation: A practical game-theoretic approach. In *The Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09)*, pages 915–922, May 2009.

[3] F. M. Delle Fave, A. Rogers, Z. Xu, S. Sukkarieh, and N. R. Jennings. Deploying the max-sum algorithm for coordination and task allocation of unmanned aerial vehicles for live aerial imagery collection. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 469–476, May 2012.

[4] F. dos Santos and A. Bazzan. Towards efficient multiagent task allocation in the robocup rescue: a biologically-inspired approach. *Autonomous Agents and Multi-Agent Systems*, 22:465–486, 2011.

[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 Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 639–646, 2008.

[6] S. Fitzpatrick and L. Meetrens. *Distributed Sensor Networks A multiagent perspective*, chapter Distributed Coordination through Anarchic Optimization, pages 257–293. Kluwer Academic, 2003.

[7] W. Ketter and A. Symeonidis. Competitive benchmarking: Lessons learned from the trading agent competition. *AI Magazine*, 33(2):103, 2012.

[8] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. RoboCup Rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *IEEE Conf. on Man, Systems, and Cybernetics(SMC-99)*, 1999.

[9] A. Kleiner, M. Brenner, T. Bräuer, C. Dornhege, M. Göbelbecker, M. Luber, J. Prediger, J. Stückler, and B. Nebel. Successful search and rescue in simulated disaster areas. In A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, editors, *Robocup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*, pages 323–334. Springer, 2005.

[10] K. Mariam, R. Paul, H. Mike, C. L. Shawn, and W. D. G. Chris. Flame simulating large populations of agents on parallel platforms. In *9th International Conference on Autonomous Agents and Multiagent Systems*, pages 10–14, 2010.

[11] A. Meisels, I. Razgon, E. Kaplansky, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *Proc. AAMAS-2002 Workshop on Distributed Constraint Reasoning DCR*, pages 86–93, July 2002.

[12] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1âĂŞ2):149–180, 2005.

[13] R. Nair, T. Ito, M. Tambe, and S. Marsella. Task allocation in robocup rescue simulation domain. In *Proceedings of the International Symposium on RoboCup*, 2002.

[14] S. Ramchurn, A. Farinelli, K. Macarthur, and N. Jennings. Decentralized coordination in robocup rescue. *The Computer Journal*, 53(9):1447–1461, 2010.

[15] S. D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. 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, 2010.

[16] C. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Proc. SIGGRAPH '87*, pages 25–34, 1987.

[17] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison Wesley, 2010.

[18] P. Scerri, A. Farinelli, S. Okamoto, and M. Tambe. Allocating tasks in extreme teams. In *Proc. of AAMAS 05*, pages 727–734, Utrecht, Netherland, 2005.

[19] N. Schurr and M. Tambe. Using multi-agent teams to improve the training of incident commanders. *Defence Industry Applications of Autonomous Agents and Multi-Agent Systems*, pages 151–166, 2008.

[20] C. Skinner and S. Ramchurn. The robocup rescue simulation platform. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1647–1648. International Foundation for Autonomous Agents and Multiagent Systems, 2010.