Decentralised Dynamic Task Allocation: A Practical Game–Theoretic Approach[*]

Archie C. Chapman
Electronics & Computer Science
University of Southampton
Southampton, SO17 1BJ, UK.
ac05r@ecs.soton.ac.uk

Rosa Anna Micillo
Department of Information Engineering
Second University of Naples
Aversa (CE), Italy
rosaanna.micillo@unina2.it

Ramachandra Kota & Nicholas R. Jennings
Electronics & Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
{rck05r,nrj}@ecs.soton.ac.uk

ABSTRACT

This paper reports on a novel decentralised technique for planning agent schedules in dynamic task allocation problems. Specifically, we use a Markov game formulation of these problems for tasks with varying hard deadlines and processing requirements. We then introduce a new technique for approximating this game using a series of static *potential games*, before detailing a decentralised solution method for the approximating games that uses the Distributed Stochastic Algorithm. Finally, we discuss an implementation of our approach to a task allocation problem in the RoboCup Rescue disaster management simulator. Our results show that our technique performs comparably to a centralised task scheduler (within 6% on average), and also, unlike its centralised counterpart, it is robust to restrictions on the agents' communication and observation range.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Scheduling; I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms, Experimentation

Keywords

Multi-agent planning, Game theory

1. INTRODUCTION

Controlling and managing the allocation of a stochastic set of tasks to a group of agents in real–time is a major challenge in the field of multi–agent systems. Relevant examples include the allocation of a city's emergency response vehicles to unfolding incidents during a major disaster (our main application focus in this work), maintenance planning problems such as aero–engine overhaul scheduling [14], and weapon-target assignment problems [1]. In all of these situations, and many others besides, robustness is a key requirement of the allocation procedure. This means we want to avoid a centralised allocation mechanism, as this could represent a single point of failure. For this reason, we focus on the use of a distributed approach in which the individual agents negotiate directly with each other to make the allocations. More specifically, these autonomous agents can operate on their own when the scenario demands this of them, or can collaborate and form teams if it is in their mutual benefit.

Now, in many cooperative settings, such as those listed above, the system designer can specify the agents' payoff for actions directly. The designer's problem, then, is to define the agents' payoffs such that their local best choice of action (given the information available to them) is one that produces a high global utility. Nonetheless, this can still leave the agents with the very difficult problem of searching a factorial–sized space for an optimal strategy in a stochastic environment containing other agents. Naturally, this can adversely affect the timeliness of the solutions produced. As such, when defining the agents' utility functions, the system designer is faced with trade–offs between the robustness of the solution mechanism, the quality of the solutions it produces, and its tractability. Within this space, we aim to develop an allocation technique that is, primarily, robust (i.e. decentralised), and secondly, tractable, to ensure solutions can be produced in a timely manner, and finally, efficient, in terms of maximising the solution quality.

Against this background, the scheduling problems that we address consist of a set of agents assigning themselves to, and executing, a dynamic stream of tasks, without the aid of a centralised manager. Each task has a hard deadline and a particular processing requirement (e.g. a casualty must be taken to hospital by 10:30am and this will take 30 minutes). In particular, the task set is dynamic because it is gradually revealed over time. This, in turn, means we require a practical mechanism that can respond to such changes in a timely fashion. Now, an agent can attend any task — there are no subtasks or specialisations — but it can act on only one task at a time and it performs this task at a fixed rate of processing. As a consequence, some of the tasks cannot be completed by an individual agent before their deadlines, and so must be tackled by a team working together (additional agents only help complete a task — they do not hinder each other). This induces a problem of scarcity among competing tasks for limited agents. Moreover, as the full set of tasks is not known at the outset, an agent has to continually negotiate with other agents over the sequence of tasks to execute, with the joint goals of completing as many as possible and in the least amount of time. Furthermore, the execution of some tasks precludes the execution of others, and decisions made at a given time affect the structure of the problem facing the agents in the future. Consequently, agents must consider the future effects of their current actions during their negotiations (e.g. the decision to rescue a particular casualty now may preclude the rescue of a second, but may not prevent the rescue of a third later).

To achieve all of these objectives, we pursue a decentralised

[*]This research was undertaken as part of the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) project jointly funded by a BAE Systems and EPSRC (Engineering and Physical Sciences Research Council) strategic partnership (EP/C548051/1) and the Hyperion project funded by DTC. The authors would like to thank Sebastian Stein for his useful comments.

Cite as: Decentralised Dynamic Task Allocation: A Practical Game–Theoretic Approach, Archie C. Chapman, Rosa Anna Micillo, Ramachandra Kota, Nicholas R. Jennings, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009),* Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. 915–922

game–theoretic approach, in which planning is achieved via negotiation between agents. In particular, we tackle this problem by first defining a global utility function, which is constructed as if we were defining it for a centralised Markov decision problem (MDP). Then, in order to develop a solution method that provides us with the robustness we require, we give control over the variables of the problem — the resources used to complete the tasks — to a set of agents, each of which makes its own independent choices. As a consequence, we use game theory to analyse their joint behaviour, because it is the natural way to model the interaction of such autonomous agents. Specifically, if the variables of a MDP are controlled by a set of autonomous agents, then we are in the realms of *Markov games* [8]. In this type of game, self–interested agents play a multi–stage game, in which the stage game varies probabilistically at each time–step as a function of the state and actions taken in the previous time–step. Here, each stage game corresponds to a particular set of tasks to be completed at that time and the status of the agents in the system (their locations, commitments, etc). Each agent has its own private utility function which it aims to maximise. However, as discussed above, these are defined such that, for any unilateral switch in strategy, an agent's change in payoff is equal to the change in the global utility. Consequently, the global maximum is a Nash equilibrium (i.e. it is a stable solution to the game). In this way, selfish agents can be used to solve an inherently cooperative problem, because their self–interest drives them towards solutions with higher global utility.

Although our problem can be cast elegantly in such a model, it is not practical to solve it at real–world scales, because agents have to consider the utilities of the future tasks present in all possible future states during the negotiations. In particular, any multi–stage game with stochastic state transitions is NEXP-complete [4], due to the factorial growth in the number of state–action–transition combinations. In order to address this concern, we show that the Markov game can be approximated using a sequence of finite length multi–stage games of complete information. In this context, we approximate the global utility function with a limited horizon version of the same function. This approximation incurs two penalties: (i) for truncating the length of the decision window and therefore ignoring payoffs from future states, and (ii) for ignoring the possible future changes in the state of the world, other than those brought about by the agents' own actions. Our use of this approximation is predicated on the assumption that changes in the world do not significantly affect the long–run payoffs to the agents; that is, all states are *quiescent*. This assumption makes sense in our setting, because the effect of scarcity of agents to complete tasks means that the introduction of additional tasks into the system only effects the agents' payoffs at the margin, and does not alter the utility of those tasks the agents may have already begun to process (or chosen not to process). Furthermore, we derive the agents' utilities from the approximate global utility function such that the agents play a *potential game* [9] at each time-step. This is very useful because, first, the maximum approximate global utility is a Nash equilibrium, and second, it implies that each game can be solved by a distributed local search algorithm. In particular, we use the Distributed Stochastic Algorithm [18] to solve each approximating game (we could equally well use alternative methods, such as Distributed Simulated Annealing or Fictitious Play, as shown in [2]).

Beyond this, we analyse the efficacy of our approach in situations where the agents' communication and observation ranges are restricted. That is, the agents cannot see the entire state of the world, and therefore must make their decision on the basis of incomplete information. This type of limitation is common in many scenarios, particularly those that possess a spatial dimension. For example, in a disaster response setting, the central message distributor may be out of action or damage to infrastructure may remove the ability to use wide–area broadcast communication. In this vein, we evaluate our technique on the ambulance–to–civilian allocation problem in RoboCup Rescue (RCR), which is an example of a spatial task allocation problem with hard deadlines and varying processing requirements. By so doing, we show that our technique performs comparably to a centralised task scheduler when communication is not limited, and that it outperforms the centralised approach as the agents' communication and observation ranges are restricted.

Given this context, this work extends the state of the art in the following ways:

1. We introduce a new technique for approximating Markov games using a series of overlapping potential games.
2. We develop a novel distributed solution technique for the approximating games, based on the Distributed Stochastic Algorithm.
3. We show that our technique is robust to restrictions on the range over which agents can communicate and observe, restrictions which typically cause centralised approaches to fail.

The paper is organised as follows. In the next section we review other approaches to distributed dynamic scheduling, and argue why they do not meet our requirements. Section 3 then introduces the game–theoretic background to our model. In Section 4 we formulate the problem as a Markov game, and describe our approximation of the global utility function. Building on this, we show how to derive agents' utilities so that the resulting game is a potential game, and describe a local search algorithm that can be used to solve it. Finally, we discuss the effects of restricting the range over which agents can communicate. Then, in Section 5, we evaluate our technique in the ambulance–civilian allocation problem in RCR. This demonstrates the benefit of using a decentralised method of control in settings where communication and observation are limited. Section 6 concludes.

2. RELATED WORK

Approaches to dynamic task allocation include: (i) domain–specific heuristics, (ii) modelling scheduling as a constraint program and solving this with either a centralised or decentralised algorithm, and (iii) auction allocation mechanisms and more general market–based approaches. Each of these will now be dealt with in turn.

First, there is a long history of using heuristics to solve scheduling problems. In particular, [13] addresses the family of *earliest deadline first* heuristic algorithms for scheduling in real–time scenarios comprising tasks with hard deadlines that compete for the same resources. The problem that we tackle here falls into this general class of problems, and, furthermore, the greedy algorithms we use as experimental benchmarks in Section 5 are also based on such heuristics. However, such algorithms rely on the centralisation of information and decision–making, and so are not appropriate for our application domains.

Second, a number of optimal algorithms for multi–agent scheduling problems have been proposed that work by reducing the problem to a constraint program. Examples of constraints include *resource relations*, which are shared by tasks that compete for the same resource, and *precedence relations*, which are shared by subtasks that need to be completed in a certain order. From these relations, a constraint program is constructed. This can be solved centrally (as in [15]) or using a decentralised constraint programming algorithm (such as DPOP [10]). Again, we rule out using centralised constraint solvers: moreover, the distributed algorithms suffer from exponential growth of some aspect of the solution process (e.g. the size of the messages passed in DPOP is exponential in the depth of the communication network it is run on), and so cannot easily be applied at the scales or on the timeframes we require.

Third, auctions and other market mechanism are beginning to be used to add robustness to task allocation by giving agents the autonomy to construct their own bids, based on their own private or partial knowledge [16, 6]. However, such auctions often rely on significant communication overhead, which can impact on the timeliness of their solutions, and, to some degree, an auctioneer represents a single point of failure (just like a central decision maker). Other market–based allocation methods, such as bargaining and exchange markets, are similar to our work, as the local search algorithm we employ to solve each potential game effectively specifies a negotiation protocol. However, our method differs from this literature, because we are able to directly specify agents' utility functions.[1]

From this landscape, the work most similar to ours is [1], in which an autonomous vehicle–target assignment problem is addressed using a potential game formulation. In their model, vehicles (agents) operate individually to optimise a global utility. The global utility optimisation is obtained via an appropriate design of the vehicles' utilities and their negotiation protocol. While we use a similar approach, there are two fundamental differences. First, in their work, vehicles are assigned to a single target, whereas in our scenario each agent is required to perform a sequence of tasks, each of which has a hard deadline. This means that our agents are required to reason over the order in which they attend to tasks, not just which tasks to attend. Second, their approach assumes that all tasks are known at the start, while we assume that they are continually discovered at run–time.

Finally, our approach to approximating the Markov game is motivated by a somewhat similar technique for producing approximate solutions to partially–observable stochastic games using a series of smaller Bayesian games [3]. In that work, a tractable Bayesian game is constructed at each time step from the most likely current states and state–transitions given an agent's beliefs. This Bayesian game is then solved to obtain a one–step policy that approximates the globally optimal solution of the original partially–observable stochastic game. In contrast, we construct a tractable multi–stage game of complete information at each time step, and because this is a potential game, it can be easily solved using a decentralised algorithm. The solution to this game is then used as a multiple–step policy to approximate the globally optimal solution.

3. PRELIMINARIES

This section introduces the foundations of our model, beginning with noncooperative games, extending these ideas to Markov games, and finally considering the class of potential games.

3.1 Noncooperative Games

A noncooperative game, $\Gamma = \langle N, \{S_i, u_i\}_{i\in N}\rangle$, consists of a set of *agents*, $N = \{1, \dots, n\}$, and for each agent $i \in N$, a set of *strategies* S_i, and a *utility function* $u_i : S \to \mathbb{R}$. A joint strategy profile $s \in S$ is referred to as an *outcome* of the game, where $S = \cup_{i=1}^{N} S_i$ is the set of all possible outcomes. An agent's utility function ranks its preferences over outcomes in terms of the payoff it receives for an outcome, such that $u_i(s) > u_i(s')$ if and only if the agent prefers outcome s to outcome s'. We will often use the notation $s = \{s_i, s_{-i}\}$, where s_{-i} is the complimentary set of s_i.

In noncooperative games, an agent's goal is to maximise its own payoff, conditional on the choices of its opponents. Stable points in such a system are characterised by the set of *Nash equilibria*. A joint strategy, s^*, such that no individual agent has an incentive to change to a different strategy, is a Nash equilibrium, i.e.:

$$u_i(s_i^*, s_{-i}^*) - u_i(s_i, s_{-i}^*) \geq 0 \quad \forall\, s_i,\ \forall\, i. \tag{1}$$

In the next subsection, we discuss an extension of this simple static game model to stochastic, multi–stage settings.

3.2 Markov Games

Markov games are an extention of standard noncooperative games, for repeated interactions in which the game played by the agents at each time–step, t, varies probabilistically as a function of the state and the choice of strategies in the previous round, or simply as some environmental process evolves [8]. Formally, a Markov game is a tuple, $\Gamma = \langle N, \Omega, \{\{S_i, u_i\}_{i\in N}\}_{\omega\in\Omega}, q\rangle$, comprising a set of *agents* $N = \{1, \dots, n\}$, a set of *state variables* $\omega \in \Omega$, a set of *stage games* $\gamma(\omega)$ indexed by elements of Ω, with each having a *strategy space* S^{ω} and a set of *utility functions* $u_i^{\omega}(s)$, defined as in the standard noncooperative model above, and a *state transition function* $q(\omega^{t+1}|\omega^t, s^t)$. The state transition function gives the probability that the next period's state is ω^{t+1}, given the current state ω^t and the strategy chosen by the agents at time t, s^t. Although state transitions are stochastic, agents are assumed to know with certainty the current state. Intuitively, payoffs in the current stage game depend only on the state and the agents' current strategies, while the probability distribution on the following state is completely determined by the current state and strategy selection. A strategy in a Markov game comprises a strategy for each of the stage games $s_i = \{s_i^{\omega}\}_{\omega\in\Omega}$; that is, there is a strategy component for every possible state of the world. Strategies in finite time step Markov games are typically evaluated by their expected total reward:[2] $\mathbf{E}[u_i(s_i, s_{-i})] = \sum_{t=0}^{T} u_i^{\omega}(s_i, s_{-i})$.

3.3 Potential Games

Potential games are a subclass of noncooperative games. They are characterised as those games that admit a potential function, which is a real-valued function on the joint strategy space whose gradient is the gradient of the constituents' private utility functions [9]. The class of finite potential games have long been used to model congestion problems on networks [11], and, recently, they have been used to analyse distributed methods of solving target assignment problems [1, 7] and job scheduling [18].

Formally, a function $P : S \to \mathbb{R}$ is a *potential* for Γ if:

$$P(s_i, s_{-i}) - P(s_i', s_{-i}) = u_i(s_i, s_{-i}) - u_i(s_i', s_{-i}) \;\forall\, s_i,\ s_i' \in S_i \;\forall\, i \in N.$$

Γ is called a *potential game* if it admits a potential. Intuitively, a potential is a function of action profiles such that the difference in its value induced by a unilateral deviation equals the change in the deviating agent's payoff.

The usefulness of potential games lies in the fact that the existence of a potential means that the game possesses two particularly desirable properties. The first is that every finite potential game possesses at least one pure strategy equilibrium [9]. Now, pure strategy Nash equilibria are particularly desirable in decentralised agent-based systems, as they imply a stable, unique outcome. Mixed strategy equilibria, on the other hand, imply a probabilistically stable, but stochastically variable equilibrium strategy profile. The second desirable property possessed by potential games is that they have the *finite improvement property*, meaning that any sequence of unilaterally improving moves converges to a Nash equilibrium in finite time. This property is important as it is used to guarantee the convergence of many simple adaptive processes to Nash equilibria in potential games (including the Distributed Stochastic Algorithm, as discussed in Section 4.3).

[1] In contrast, market–based task allocation methods are designed to incentivise agents with arbitrary utility functions to act in a way that maximises a social welfare function. Nonetheless, the connections between our work and mechanism design, in particular the *Groves mechanism*, are discussed in Section 4.3.

[2] This is in contrast to infinite time step Markov games, which typically use the discounted expected total reward.

4. THE TASK ASSIGNMENT MODEL

We begin this section by defining our task allocation problem as a Markov game. In 4.2 we describe our finite–horizon approximation of the global utility function, and in 4.3 we show how agent utility functions are derived so that the agents play a potential game. Section 4.4 then discusses the Distributed Stochastic Algorithm, which we use to solve the approximating potential games. Finally, in 4.5 we discuss the effects on our approach of limiting the distance over which agents can observe or communicate.

4.1 Markov Game Formulation

The full task allocation model is a finite Markov game of complete information: the current state is known, future states are uncertain, agents have a finite set of strategies and play for a finite number of time steps. As per Section 3.2, our model comprises:

- A set of *states* $\omega \in \Omega$, each of which defines a set of *tasks* $X = \{x_1, x_2, \dots x_j, \dots\}$, with each task possessing a *deadline*, $t^d_{x_j}$, a number of required *processing units*, y_{x_j}, and a *task utility function*, $u_{x_j}(s) : S \to \mathbb{R}$,
- A set of *agents* $N = \{1, 2, \dots, i, \dots, n\}$, each with a *strategy space* S_i, with elements s_i, composed of a sequence of tasks to attend, and an *agent utility function* $u_i(s_i, s_{-i}) : S \to \mathbb{R}$,
- A state transition function $q(\omega^{t+1}|\omega^t)$, and
- A *global utility function* $u_g(s) : S \to \mathbb{R}$.

The problem we face is to design the agents' utility functions and a distributed negotiation protocol such that the system produces high quality solutions. The other elements of the model — the transition function and the task and global utility functions — come directly from the problem specification. To begin, the transition function describes how new tasks are generated and introduced into the system. Then, the task utility function represents the payoff for completing a task, and in this case it is:

$$u_{x_j}(s) = \left\{ \begin{array}{ll} \beta^{t^c_{x_j}(s)} & \text{if } t^c_{x_j}(s) \le t^d_{x_j} \\ 0 & \text{otherwise,} \end{array} \right\} \tag{2}$$

where $t^c_{x_j}(s)$ is the completion time for the task, given the agents' strategies s, $t^d_{x_j}$ is the hard deadline for successfully completing a task, and $0 < \beta \le 1$ is a discount factor that incorporates any benefit of completing the task earlier. In more detail, the task utility function is designed to possess two properties important in our scenario. First, the conditional statement models the hard deadline, so if fewer than the minimum number of agents required to complete the task before $t^d_{x_j}$ attend to it, the utility is zero, even if some agents do attend to the task. This is important because, in our scenarios, tasks incomplete at deadline are equivalent to unattended tasks. Second, increasing the number of agents beyond the number necessary to complete the task by its deadline improves the completion time, which raises the task payoff. This captures the benefit of completing tasks earlier.[3] Finally, the global utility function ranks the overall allocation of tasks to agents, and is an aggregation of task utilities:

$$u_g(s) = \sum_{x_j \in X} u_{x_j}(s). \tag{3}$$

This preserves the desirable properties of the task utility function.

[3]The value of β in Equation 2 represents a trade–off between the number of tasks completed and the timeliness of those completed tasks. As we aim to maximise the number of tasks completed, we chose a value close to 1, however, if timeliness was our main concern, we would choose a lower value.

Now that we have defined the task and global utility functions for our problem, if we were working directly with the Markov game model, we would define the agents' utility functions. However, note that an agent's strategy space in this model is the set of all permutations of assignments to tasks each period; a strategy prescribes an action for each time step for every contingent state of the world. Thus, an agent's strategy is a set of vectors of actions, one vector for each state of the world, with an agent's utility function defined over this set. Given the huge number of possible states and action vectors of extremely large sizes (factorial on the number of tasks), evaluating and negotiating a set of joint strategies for this problem is clearly a very involved process, that would likely take a great deal of time. Furthermore, given that we intend to deploy our agents in a system where they will be required to make decisions over a short time horizon, constructing such a strategy for the full set of possible outcomes is practically impossible, due to the huge number of possible future states and action combinations that need to be evaluated. For these reasons, we instead derive the agents' utility functions from a tractable approximation of the global utility function for the stochastic game.

4.2 Approximate Global Utility Function

Rather than attempting to solve the Markov game above, we approximate it using a series of static potential games of complete information. Specifically, in this section we approximate the global utility function using a technique similar to a look–ahead policy commonly used in MDPs (see, for example, [12], Chapter 5). We can use this type of approximation because, as discussed in Section 1, we expect all states to be quiescent.[4] Following this, in the next subsection we derive agents' utilities from this approximate global utility, such that the agents' negotiation problem forms a potential game.

In more detail, the global utility is approximated as follows. At each time step, a game is constructed with each agent's strategy defined for a fixed decision window of w future time steps. In each of these games, an agent's strategy is a vector of tasks to attend to during the interval $[t, t+w]$, $s_i = \{x^t, x^{t+1}, \dots, x^{t+w}\}$. In this way, at each time step, the Markov game is approximated by a static game of complete information defined over the next w time steps. Then, the task utility functions in each approximating game are defined as in Equation (2), with the addition that, for tasks not completed by $t+w$, payoffs are calculated as if all agents' final strategy component s_i^{t+w} is repeated until the task in question is completed or its deadline passes. If we did not assume the continuation of these tasks, the utility of all incomplete tasks at time $t+w$ would be zero, potentially leading to an artificial bias against tasks with large processing requirements and/or long completion times. The global utility of this model is of the same form as that for the Markov game model:

$$u_g^{t,w}(s) = \sum_{x_j \in X} u_{x_j}(s), \tag{4}$$

except that the constituent task utilities are defined over the restricted interval $[t, t+w]$.

Before deriving the agents' utility functions, we discuss two types of errors introduced by our approximation of the global utility function. The first is the restriction of strategies to the decision window $[t, t+w]$. The result of this is that the value of future states beyond the next w time steps are not evaluated with the current choice of strategy. Now, although we only maximise $u_g(t)$ over the next w

[4]If any state is non–quiescent, then our use of a look–ahead style approximation will suffer from the *horizon problem*, meaning it will not be able to avoid entering states that lead, unavoidably, to bad outcomes beyond the length of the decision window used.

time steps, any adverse effects on the full $u_g(t)$ are expected to be small, because the game puts greater importance on tasks with closer deadlines. Similarly, if we used the full Markov game model, tasks with earlier deadlines would be processed earlier. The second type of error is caused by not incorporating information about the exogenous evolution of the world (in our model, the arrival of new tasks) into the choice of state. However, as argued earlier, in the domains we consider, the state of the world moves slow enough for us to ignore this effect without introducing any significant errors.

Now, because we are working on a problem for which a sequentially optimal solution is intractable, we are faced with a trade–off between the two sources of approximation error. The first type is reduced as the restriction on the size of w is relaxed. On the other hand, the second type is mitigated by using a shorter length window, because the difference in the predicted and actual state reached in the future is reduced. Consequently, our choice of window length reflects the need to balance the effect of these two concerns. In practice, this means that the value of w has to be determined experimentally as it depends on the domain (elaborated upon in Section 5.2).

4.3 Deriving the Agents' Utility Functions

Given the above approximation of the global utility function for our problem, the agent's payoffs are designed so that any increase in an agent's utility corresponds to an increase in $u_g^{t,w}(s)$. Now, because the global utility is the sum of task utilities, an agent's marginal contribution to the global utility can be specified in terms of the sum of its contributions to individual tasks — that is, the difference between the task utility when the agent contributes to the task and when it does not. This type of utility structure is called *wonderful life utility* by [17]. The marginal contribution of agent i to a task x_j is given by:

$$mu_i^{x_j}(s_i,s_{-i}) = u_{x_j}(s_i,s_{-i}) - u_{x_j}(s_0,s_{-i}), \qquad (5)$$

where s_0 is the *null strategy*, in which the agent does not contribute to completing any task. Note that this form of utility function is similar to the Groves mechanism, in which agents in a team are paid an amount equal to their marginal contribution to the team utility [5].[5] In contrast, in our setting we can do away with the explicit utility transfers that occur in mechanism design because the system designer is able to specify each agent's utility function directly (recall the discussion in Section 1). In particular, by using the wonderful life utility, a system designer internalises the effect of an agent's actions on the global utility.

The relationship between the task utility function and an agent's marginal contribution to the task utility is shown in the example in Figure 1. This shows $u_{x_j}(s)$ and $mu_i^{x_j}(s)$ for a task requiring 4 units of processing and with a deadline 2. A minimum of 2 agents are required to complete the task — a constraint captured by the increase in the task and agents' utilities as the number of agents increases from 1 to 2. If more than this number of agents attend, the task utility continues to increase as the completion time decreases, however, the marginal contribution of each additional agent beyond this point decreases.

An agent's marginal utility values are used to construct its payoff for each strategy, which is the sum of its marginal contributions to all the tasks it attends to in the next w time steps:

$$u_i(s_i,s_{-i}) = \sum_{x_j \in s_i} mu_i^{x_j}(s_i,s_{-i}) \qquad (6)$$

[5]In order to make the connections clear, observe that if we were trying to incentivise agents, who possess private preferences, to act in a certain manner, then a mechanism design procedure, such as the Groves mechanism, would be an appropriate choice of control mechanism.

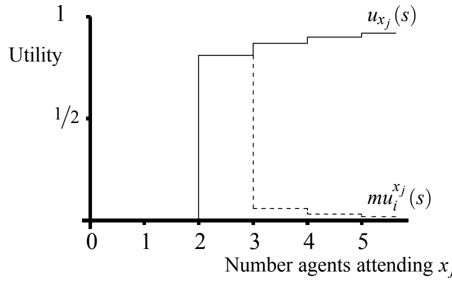

Figure 1: An example of task utility and agent marginal utility, with $y_x = 4$, $t_x^d = 2$ and $\beta = 0.9$.

Note that the first summation could be taken over all tasks in X with the same result, as $mu_i^{x_j}(s_i,s_{-i})$ is zero for all tasks to which i does not contribute. This point is important, as it implies that a change in strategy that increases an agent's utility always corresponds to an increase in the global utility restricted to the decision window $[t,t+w]$. Consider the difference in i's utility for switching from s_i to s_i'. The following shows that the change in an agent's own utility is equal to the change in the global utility:

$$\begin{aligned}
&u_i(s_i,s_{-i}) - u_i(s_i',s_{-i}) \\
&= \sum_{x_j \in s_i}\Big(u_{x_j}(s_i,s_{-i}) - u_{x_j}(s_0,s_{-i})\Big) - \sum_{x_j \in s_i'}\Big(u_{x_j}(s_i',s_{-i}) - u_{x_j}(s_0,s_{-i})\Big) \\
&= \sum_{x_j \in X}\Big(u_{x_j}(s_i,s_{-i}) - u_{x_j}(s_0,s_{-i}) - u_{x_j}(s_i',s_{-i}) + u_{x_j}(s_0,s_{-i})\Big) \\
&= u_g^{t,w}(s_i,s_{-i}) - u_g^{t,w}(s_i',s_{-i}).
\end{aligned}$$

Thus, a game played between agents with utility functions as given above is a potential game, with a potential function given by the global utility function over the decision window. There are two consequences to this result. First, the globally optimal allocation of tasks to agents in the window resides in the set of Nash equilibria. To see this, assume that the optimal point is not a Nash equilibrium. Then there must be some agent that can alter its state to improve its utility, which, in turn, will improve the global utility, which contradicts the assumption that the optimal point is not a Nash equilibrium. Despite that, in most cases some sub–optimal Nash equilibria also exist. Second, the game has the finite improvement property (see Section 3.3), implying the convergence of the Distributed Stochastic Algorithm, as we discuss in the next section.

4.4 The Distributed Stochastic Algorithm

The Distributed Stochastic Algorithm (DSA) is a greedy local search algorithm [18]. We use it here because previous work comparing the performance of simple distributed heuristics in the closely related class of distributed constraint optimisation problems has identified it as an algorithm that can quickly produce good quality solutions with a low communication overhead [2].

In more detail, DSA is a synchronous algorithm, in that agents act in step, however, at each time step, an agent has some probability p of activation, known as the degree of parallel executions [18]. Given an opportunity to update its action, an agent selects the action with the greatest increase in payoff — its best response — or if no change improves the payoff, the agent does not change its strategy. The algorithm is motivated by observing that when neighbouring agents adapt their strategies at the same time, they can inadvertently move their joint state to a globally inferior outcome, a phenomenon known as 'thrashing'. Although the probabilistically parallel execution of agents' updating procedures does not prevent all thrashing from taking place, by selecting an appropriate value of p, thrashing behaviour can be minimised. Importantly, DSA converges to a Nash equilibrium in potential games. Briefly, this is because no agent will leave a Nash equilibrium after the first time

$$\begin{tikzcd}
\textbf{t:} & s^t & = \big[& s_i^t & s_i^{t+1} \arrow[dl] & s_i^{t+2} \arrow[dl] & \cdots \arrow[dl] & s_i^{t+w} \arrow[dl] & \big] \\
\textbf{t+1:} & s^{t+1} & = \big[& s_i^{t+1} & s_i^{t+2} \arrow[dl] & s_i^{t+3} \arrow[dl] & \cdots \arrow[dl] & s_i^{t+1+w} \arrow[dl] & \big] \\
\textbf{t+2:} & s^{t+2} & = \big[& s_i^{t+2} & s_i^{t+3} & s_i^{t+4} & \cdots & s_i^{t+2+w} & \big]
\end{tikzcd}$$

Figure 2: Recycling solutions: Typically, the $t+1$ to $t+w$ strategy components from game t are used as initial conditions for DSA in game $t+1$, and so on.

it is played, and for values of $0 < p < 1$, the probability that the agents play a Nash equilibrium goes to 1 as time progresses, as a consequence of the finite improvement property.

In application domains with short decision horizons, like RCR, a good initial set of conditions can significantly improve the convergence time of DSA. For this reason, in our model, solutions to one approximating game are used as initial conditions, or partial solutions, for DSA in the following game. Specifically, because the $t+1$ to $t+w$ strategy components of consecutive games overlap, we can reuse these components as the starting conditions for DSA in each subsequent game (as shown in Figure 2). This is particularly useful for situations where the number of negotiation steps is limited by communication restrictions (such as in RCR). A related issue is that newly arriving tasks with pressing deadlines have the potential to induce significant re–coordination by the agents, rendering the partial solutions negotiated in previous game's strategies irrelevant. In this way, these new tasks have the potential to disrupt the convergence of the algorithm. However, in practice, by using any reasonable value for p (i.e. $0 \ll p \gg 1$) in conjunction with a long window, the agents are able to deal with such disruptions in a graceful manner. That is, as long as p imparts significant inertia on the existing strategy, it will prevent the algorithm from being significantly disrupted.

4.5 Handling Limited Range Communication

So far in this section, we have showed how to implement a general distributed technique for solving a dynamic task allocation problem. However, as motivated earlier, we also wish to develop a technique that is robust in the face of restrictions on the distance over which agents can communicate. In particular, the technique we develop is a natural extension to the wonderful life utility described in Section 4.3, and is appropriate for any task allocation problem with a spatial dimension. In more detail, we model situations where an agent can communicate over a limited distance, r, and is only aware of some of the tasks that are currently know about in the system. As such, the major changes in the method are that: (i) the set of strategy components available to an agent is restricted to only those tasks it is aware of, X_i,[6] and (ii) the agent's utility computations are carried out using only the strategies of those agents that are currently within its communication range, $j \in N_i$. This gives us the following agent utility functions:

$$u_i(s_i, s_{N_i}) = \sum_{x_j \in s_i} mu_i^{x_j}(s_i, s_{N_i}) = \sum_{x_j \in s_i} \left(u_{x_j}(s_i, s_{N_i}) - u_{x_j}(s_0, s_{N_i}) \right), \quad (7)$$

where s_i is restricted to the set of tasks i is aware of, X_i. Now, using this form for the agents' utility functions means that the approximate global utility function (Equation 4) need not be a potential function for the game. However, if all agents are aware of those agents attending to their tasks, then Equation 4 acts as a potential function. This is the case when the agents are at (or sufficiently near) the location of their tasks. On the other hand, because the components of an agent's strategy are restricted to those tasks it is aware of, parts of the global utility function are, in effect, inaccessible to the agents. Nonetheless, the accessible local maxima of the approximate global utility are Nash equilibria of the game.

5. APPLICATION TO ROBOCUP RESCUE

In this section, we describe an application of our technique to RCR, which is a simulation of a disaster response scenario.[7] RCR is chosen because it is a well–known domain used for benchmarking multi-agent coordination algorithms. It is a complex problem in which teams of agents have to allocate and perform tasks using incomplete information in a stochastic environment, in real time. Thus, it provides an ideal platform for evaluating the efficacy of our model. We begin by mapping from RCR to our generic model, before discussing the experiments we use to evaluate our approach.

5.1 Ambulance–Civilian Allocation

A major part of the RCR problem is the allocation of injured civilians to ambulances. We characterise this aspect of RCR using our model, as described in Section 4, where each agent, i, corresponds to an ambulance and each task x_j represents an injured civilian that needs rescue. As in our model, each civilian has a hard deadline, $t_{x_j}^d$, by which time it must reach a hospital if it is to survive, and a processing requirement, y_{x_j}, corresponding to the time it would take a single ambulance to be in a position to remove it from the scene. In RCR, the number of injured civilians is typically much greater than the number of ambulances, and not all of them are known to the ambulances at the start. Rather, they are discovered over time. This means that an ambulance must negotiate a sequence of civilians to attend to with other ambulances, with the rescue of some civilians requiring more than one ambulance because of a high y_{x_j} and/or an imminent $t_{x_j}^d$.

Given this, the task completion time, $t_{x_j}^c(s)$ in Equation 2, is the time it takes a team of ambulances, given by the joint strategy profile s, to rescue the civilian. This incorporates both the civilian's processing requirement (the time needed by the team before taking it to the hospital), as well as estimates of the time it takes for the team members to travel to the civilian. The global utility $u_g(s)$ increases with the number of civilians rescued, and for each civilian, increases with lower completion times. Regarding an ambulance's marginal utility (Equation 5), because β is close to 1, the contribution of an ambulance that is critical to the rescue of civilian x_j before $t_{x_j}^d$ is greater than the benefit of speeding up the rescue of a civilian that is already assured of being saved. This effect is demonstrated in Figure 1. Following this, an agent's utility (Equation 6) is then the sum of its contribution to all tasks in the window $[t, t+w]$, and consequently, the approximate global utility function acts as a potential for the entire game. Thus, the salient features of this problem are captured in our model.

Nonetheless, two small variations to the standard DSA are necessary to successfully implement our model in RCR. First, one component of an ambulance's role is to transport rescued civilians to a hospital, which takes time and can upset the agent's strategy. Because of the difficulties of capturing this requirement in our model, we allow the following innovation to DSA: Whenever an ambulance has returned a civilian to a hospital, it computes a completely new best strategy. Second, because agents are not computing their best strategy at every time step, if others have changed their strategy, it is possible that an agent's value for a task completion time differs significantly from the true value. This may require the agent to shift forward many elements of its strategy vector (many more than illustrated in Figure 2). Now, if too many elements are re-

[6]The way that agents learn about tasks is typically specific to the domain, and how this occurs in RCR is discussed in Section 5.2.

[7]`http://www.robocuprescue.org`

moved, the resulting recycling may be counter–productive. For this reason, if, in recycling past components, an agent's strategy vector is advanced up by less than $w/3$ components, then with probability p the agent computes a completely new strategy, and with probability $(1-p)$ it generates a new strategy for the remaining components only, as usual. However, if more than than $w/3$ components are removed, the agent always computes an entirely new strategy.[8]

5.2 Experimental Design

We now discuss the evaluation of our overlapping potential game approximation (OPGA) algorithm in RCR. Specifically, we ran OPGA on three standard RCR maps — Kobe, VC (Virtual City) and Foligno, each containing 80 civilians and 6 ambulances. Foligno is a larger and less structured map than Kobe or VC, making it harder to detect civilians there, while in the Kobe scenario, new casualties are revealed at a quicker rate than in the others. Information about casualties is gathered by fire–brigades and police–patrols that explore the map and pass it on to the ambulances, thus simulating a dynamic continuous stream of tasks. In the limited range scenario, police and fire–brigades can only pass on information to an ambulance when they are within its communication range, thus representing the limited observation range of the ambulances. Each simulation is 300 time steps in duration. We implement two parameter setting of our method with $p = 0.5$ or 0.9 — OPGA(0.5) and OPGA(0.9) — and a decision window of $w = 30$ steps for both. As our preliminary experiments showed that the results were not very sensitive to different p values between 0.5 and 0.9, we limited our current results to the two endpoints of this range. The value for w was narrowed down through experimentation, as it depends on the nature of the domain. To achieve statistical significance (results are shown with 95% confidence intervals), each experiment was run 30 times. The performance in an experiment is reflected by the score obtained at the end of the simulation. This score is a standard provided by the RCR framework and is basically a sum of the health points of the civilians in the map. The health of an unrescued civilian decreases with time until it reaches 0 (which, in fact constitutes the deadline), while that of a rescued civilian improves with time. We also measure the number of civilians saved over time, because it gives an insight into how the rate of rescue is affected by the rate of discovery of casualties.

We ran two batches of experiments. In the first, the agents' communication range was not restricted. We use these results to directly compare the performance of OPGA to a centralised greedy (myopic) scheduler as an upper bound and a decentralised greedy heuristic as a comparable lower benchmark. These benchmarks both use the earliest dealine first heuristic, disussed in Section 2, to allocate agents to tasks. The former lists civilians in order of their deadline, centrally allocates free ambulances to the civilian with the earliest deadline up to the point where it is assured of being completed, and then allocates ambulances to the next civilian on its list, and so forth. Now, because the allocation is performed centrally, no mis–coordination is possible — neither fewer, nor more agents than are required will ever be allocated to a civilian. As such, this scheduler should out perform OPGA. Under the decentralised greedy heuristic, each ambulance simply attends to the task with the shortest deadline. This approach will typically lead to an over–allocation of ambulances to civilians with short deadlines, and it will occasionally allocate ambulances to a civilian even when their efforts to save it are bound to be futile.

In the second batch of experiments, we test the performance of OPGA with restrictions on the range of agents' communication and observations, as discussed in Section 4.5. These restrictions are 20% 15%, 10%, and 5% of the maximum distance between two points on the map in question (so the area covered by an agent's range decreases quadratically with these values). In this batch, we compare OPGA(0.9) to the centralised greedy scheduler only. We do this to test our hypothesis that OPGA performs better than the centralised scheduler in scenarios where the communication and observation range is restricted.

[8]Experimental evidence has shown two–thirds to be a reasonable value for this threshold.

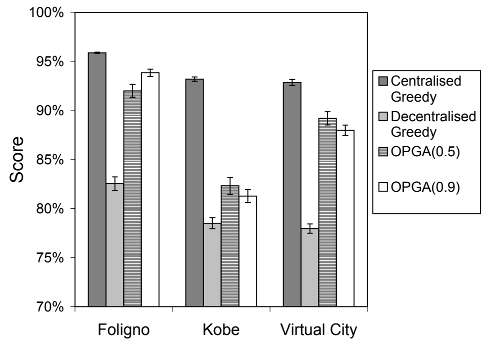

Figure 3: Comparing the methods across three maps.

5.3 Results

To begin with, we discuss the results of the first batch of experiments: Figure 3 show the mean performance of OPGA(0.9) and OPGA(0.5) compared to the centralised and decentralised greedy methods in the three maps. Although the difference in score between OPGA and the centralised greedy approach is statistically significant, it is only 6% worse, on average, than the centralised approach. Furthermore, OPGA performs significantly better than the decentralised greedy method. When taken together, these results show that our approach, based on overlapping potential games, is a good approximation of the optimal solution to the Markov game model of the ambulance–to–civilian problem in RCR. In more detail, both versions of OPGA perform better in the Foligno and VC scenario than in Kobe. Furthermore, a $2^k r$ factorial design test on the results evaluating the effects of the value of p and the map on the score indicates that 95% of the variation of the score is explained by variation in the map, and less than 1% by variations in p. The cause of the variation in scores between maps is due to the rate at which new trapped civilians are introduced. In particular, civilians are discovered at a quicker rate in the Kobe map than in Foligno or VC. This is illustrated clearly in Figure 4. Here, a slower rate of discovery allows OPGA to find good quality solutions more regularly than in maps where, at times, the rate of civilian discovery is faster. Thus, OPGA performs better in Foligno and VC than in Kobe. Furthermore, this matches with the assumption we make that the state of the world moves slow enough for us to ignore the effect of the possible changes to the state of the world (in particular, the list of civilians), without inducing significant errors. When this assumption is less warranted, as in the Kobe scenario, the algorithm performs relatively worse.

Now we turn to the second batch of experiments (Figure 5). Observe that the performance of the centralised greedy algorithm degrades quicker than OPGA, both in terms of its mean score and the variability in its performance (as seen in the larger error bars at each restriction level). In contrast, OPGA performs better than the centralised approach whenever the agents' communication range is restricted, with the exception of Kobe when restricted to 5%; in this case, the information flowing to the agents is minimal, hence the performance of any method will tend to be poor. This is precisely the effect of restricting the communication and observation range we expected to see, and justifies our arguments for using a principled decentralised algorithm in restricted range environments.

Furthermore, for moderate restrictions (20-15%), the performance

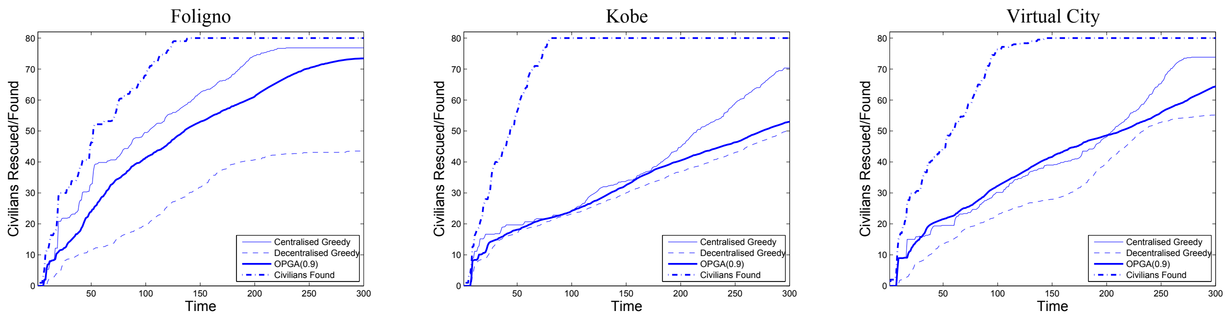

Figure 4: Comparing the methods on the Kobe, VC and Foligno maps, alongside the number of casualties found.

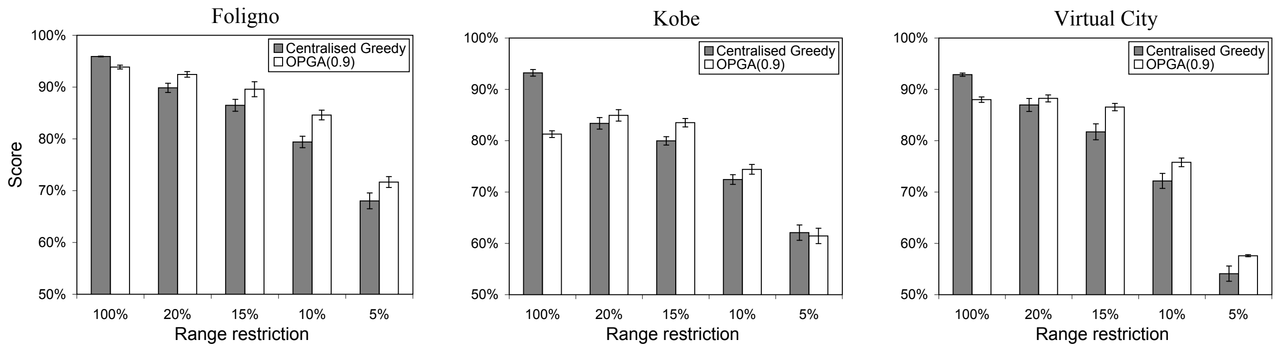

Figure 5: Comparing the methods as the communication and observation range is restricted.

of OPGA actually improves in both Kobe and VC. This is a surprising result. It occurs because, under restricted information, the quiescence assumption motivating our choice of approximation is better supported than in the full information case. That is, the state of the world, as perceived by each agent, is more stable when their communication and observation ranges are moderately restricted. The subsequent degradation in performance of OPGA is due to a simple lack of information flowing to the agents. This effect is not reproduced in Foligno because OPGA does well, even in the full communication case, as the rate of discovery of casualties is slow.

6. CONCLUSIONS

This paper defines a game–theoretic technique for decentralised planning to address dynamic task allocation problems. There are two main aspects to the problem addressed in this paper. First, each agent has to perform a sequence of tasks over time and often tasks may require more than one agent for their successful completion. Second, the set of tasks is dynamic as new ones are discovered over time. This leads to a Markov game formulation. However, as such stochastic games are generally intractable, we propose a technique for approximating such games using a sequence of potential games. Finally, we demonstrate how to apply Distributed Stochastic Algorithm to solve these approximating games. To evaluate the efficacy of our general approach we implement it in RoboCup Rescue. In so doing, we find it performs comparably to a centralised task scheduler when communication is not limited, and that it outperforms the centralised approach as the agents' communication and observation ranges are restricted.

The next step of this work is to extend our model to capture other aspects of complex disaster scenarios, such as allowing agents to have differing costs for performing the same task (e.g. agents with different levels of skill) and representing deadlines by a distribution over times (i.e. incomplete information about the tasks). The former necessitates extending the agents reasoning to cover others' types, while the latter requires an extension of tasks utilities functions to capture soft deadlines.

7. REFERENCES

[1] G. Arslan, J. R. Marden, and J. S. Shamma. Autonomous vehicle-target assignment: A game theoretical formulation. *ASME Journal of Dynamic Systems, Measurement and Control*, 129:584–596, 2007.

[2] A. C. Chapman, A. Rogers, and N. R. Jennings. Benchmarking hybrid algorithms for distributed constraint optimisation games. In *Proceedings of OptMas '08*, 2008.

[3] R. Emery-Montemerlo, G. Gordon, J. Schneider, and S. Thrun. Approximate solutions for partially observable stochastic games with common payoffs. In *AAMAS*, pages 136–143, 2004.

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

[5] T. Groves. Incentives in teams. *Econometrica*, pages 617–631, 1973.

[6] S. Koenig, C. Tovey, X. Zheng, and I. Sungur. Sequential bundle-bid single-sale auction algorithms for decentralized control. In *IJCAI*, pages 1359–1365, 2007.

[7] M. Krainin, B. An, and V. Lesser. An application of automated negotiation to distributed task allocation. In *IAT*, pages 138–145, 2007.

[8] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *ML*, pages 157–163, 1994.

[9] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

[10] B. Ottens and B. Faltings. Coordinating agent plans through distributed constraint optimization. In *ICAPS Multiagent Planning Workshop*, 2008.

[11] R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[13] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. *Deadline Scheduling for Real-time Systems*. Springer, 1998.

[14] A. Stranjak, P. S. Dutta, M. Ebden, A. Rogers, and P. Vytelingum. A multi–agent simulation system for prediction and scheduling of aero engine overhaul. In *AAMAS*, 2008.

[15] W.-J. van Hoeve, C. Gomes, M. Lombardi, and B. Selman. Optimal multi-agent scheduling with constraint programming. In *IAAI*, 2007.

[16] M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.

[17] D. H. Wolpert and K. Tumor. An overview of collective intelligence. In J. M. Bradshaw, editor, *Handbook of Agent Technology*. AAAI Press/MIT Press, Cambridge, MA, 1999.

[18] W. Zhang and Z. Xing. Distributed breakout vs. distributed stochastic: A comparative evaluation on scan scheduling. In *DCR*, pages 192–201, 2002.