Verifying Team Formation Protocols with Probabilistic Model Checking*

Taolue Chen, Marta Kwiatkowska, David Parker, and Aistis Simaitis

Computing Laboratory, University of Oxford,
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

Abstract. Multi-agent systems are an increasingly important software paradigm and in many of its applications agents cooperate to achieve a particular goal. This requires the design of efficient collaboration protocols, a typical example of which is *team formation*. In this paper, we illustrate how probabilistic model checking, a technique for formal verification of probabilistic systems, can be applied to the analysis, design and verification of such protocols. We start by analysing the performance of an existing team formation protocol modelled as a discrete-time Markov chain. Then, using a Markov decision process model, we construct optimal algorithms for team formation. Finally, we use stochastic two-player games to analyse the competitive coalitional setting, in which agents are split into cooperative and hostile classes. We present experimental results from these models using the probabilistic model checking tool PRISM, which we have extended with support for stochastic games.

1 Introduction

Multi-agent systems have become an important software paradigm. One of the key ideas behind this approach is that several different agents can cooperate to achieve certain goals. This requires the design of efficient collaboration protocols, of which *team formation* is a typical example. In this paper, we focus on a distributed team formation protocol introduced in [10]. There, the authors used it to analyse team performance in dynamic networks. The protocol has also been applied to coalition formation for data fusion in sensor networks [11]. In both cases it has been used as a basis for designing other algorithms, which makes it a compelling target for formal analysis.

The basic setting for the protocol of [10] consists of an *agent organisation*, i.e., a network of interconnected agents which have certain resources. These agents attempt to form teams in order to accomplish tasks which are generated periodically and globally advertised to the agent organisation. The topology of the network restricts the set of possible agent teams – for an agent to be on a team, the agent must have a connection with at least one other agent in that team. Tasks are generic in that they only require a team of agents with the necessary resources to accomplish the specific task. As in [10], we do not consider the solution process, but only the team formation.

As is typical for multi-agent algorithms, probabilities play a crucial role in team formation protocols. Firstly, agents are scheduled to act in a random order, following the

* This work is supported by the ERC Advanced Grant VERIWARE.

approach of [10]; secondly, in our setting, tasks are drawn from a task pool, following some known probability distribution. This is particularly interesting for the *online* version of the algorithm (see Alg. 3), where tasks are generated after teams have formed. In this case, agents have to choose strategies to optimise against a set of tasks governed by a certain probability distribution rather than a particular task. Finally, probabilities are used to implement strategies of agents themselves, for example random selection of a team to join. These issues motivate the use of analysis techniques that take can take probabilistic behaviour into account.

Formal verification is an approach to check the correctness of a system using rigorous, mathematical reasoning. Fully automated verification techniques such as *model checking* have proved to be widely applicable, including to multi-agent systems [17]. In this paper, as described above, the systems that we study exhibit *probabilistic* behaviour. Thus, we use *probabilistic model checking*, an automated technique for the formal verification of *stochastic* systems.

Probabilistic model checking is based on the construction of a probabilistic model from a precise, high-level description of a system's behaviour. The model is then analysed against one or more formally specified *quantitative* properties, usually expressed in temporal logic. These properties capture not just the *correctness* of the system, but a wide range of measures such as *reliability* or *performance*. We can compute, for example, "the probability that the algorithm successfully terminates within k rounds". By augmenting the model with *rewards*, a further range of properties can be analysed.

In addition to offering convenient high-level formalisms for representing models and their properties, the strength of probabilistic model checking is that offers *exact*, *exhaustive* analysis techniques. Rather than, for example, discrete-event simulation (as it is done for team formation protocols in [10]), probabilistic model checking is based on an exhaustive exploration and numerical solution of the model, allowing best- and worst-case behaviour to be identified. This is particularly valuable for distributed protocols (like the ones in this paper), whose behaviour is notoriously difficult to understand precisely. Furthermore, efficient techniques and tools exist for this purpose.

In this paper, we use the PRISM probabilistic model checker [16] to analyse various agent organisations for the team formation protocol of [10]. We use several different types of probabilistic models and express quantitative performance properties of them in temporal logic. Firstly, we model the original version of the protocol using discrete-time Markov chains (DTMCs), where the behaviour of each agent is described entirely in a probabilistic (deterministic) way. Then, we extend the original algorithm by allowing agents to make decisions nondeterministically, instead of randomly, when forming teams; such systems are naturally modelled by Markov decision processes (MDPs). By analysing the MDP, we obtain the best- and worst-case performance of agent organisations. MDPs, however, can only model fully collaborative behaviour, whereas in many scenarios it is crucial to address hostile behaviour of some agents in the organisation. To cater for this we use stochastic two-player games (STPGs) as a model for the system containing two groups of agents – *collaborative* and *hostile* – which try to, respectively, maximise or minimise the performance of the organisation (i.e. this effectively becomes a zero-sum stochastic two-player game). Orthogonal to these, we consider two differ-

ent settings, namely *offline* and *online*, depending on whether the tasks are generated respectively *before* and *after* teams have formed (see Alg. 3).

Our experiments illustrate several aspects of agent organisation analysis. As a typical case, we choose four network topologies, each consisting of five agents, i.e., fully connected, ring, star, and a network having one isolated agent. For each one, we compute the expected performance of the organisation and find organisation-optimal resource allocation among agents. Then we show using MDP model checking what is the best performance that can be achieved by this organisation. Lastly, we take the model to the STPG setting to obtain the optimal coalitions of different sizes and evaluate their performance. For all of these cases, we consider the offline and online dichotomy.

In summary, the main contributions of this paper are as follows:

(1) We perform a comprehensive and *formal* analysis of the performance of the team formation protocol proposed in [10].
(2) We *extend* the original algorithm of [10], allowing agents to make decisions non-deterministically when forming teams. Then, by modelling and analysing as an MDP, we *synthesise* the best strategies for agents to achieve optimal performance, partially solving an open problem posed in [10].[1]
(3) We *extend* the PRISM model checker with support for modelling and automated analysis of STPGs and *address the competitive coalitional setting*, in which agents are split into cooperative and hostile classes, using *stochastic games* to synthesise optimal agent coalitions. To the best of our knowledge, this is the first work to perform a *fully-automated* probabilistic analysis of this kind.

We note that it would be difficult to achieve (2) and (3) using simulation-based approaches; this demonstrates the strength of formal verification.

Related work. Cooperative behaviour, which is one of the greatest advantages of agent-based computing, has been studied from many different angles over the years. Coalitional games have traditionally been analysed from a game-theoretic perspective [19], but in recent years have attracted a lot of attention from researchers in artificial intelligence, especially in cooperative task completion [20]. Several approaches for team formation and collaborative task solving have been considered including team formation under uncertainty using simple heuristic rules [13], reinforcement learning techniques [1] and methods using distributed graph algorithms [18]. To reason formally about cooperative games, several logics (e.g., Alternating Time Logic [3], Coalitional Game Logic [2], Strategy Logic [6]) and other formalisms (e.g., Cooperative Boolean Games [8]) have been introduced and used to analyse coalitional behaviour [5]. Model checking has been used to analyse (non-probabilistic) knowledge-based properties of multi-agent systems, using the tool MCMAS [17]. Probabilistic model checking was employed to analyse probabilistic agents in negotiation protocols [4] (but only for fixed strategies modelled as DTMCs) and to Byzantine agreement protocols [14].

[1] We quote: "the problem of developing or learning effective team initialising and team joining policies is also important, and is included in our on-going and future work".

2 Preliminaries

2.1 Probabilistic Models

We begin with a brief introduction to the three different types of probabilistic models that we will use in this paper.

Discrete-time Markov chains (DTMCs) are the simplest of these models. A DTMC $(S, \mathbf{P})$ is defined by a set of states S and a probability transition matrix $\mathbf{P} : S \times S \to [0, 1]$, where $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ for all $s \in S$. This gives the probability $\mathbf{P}(s, s')$ that a transition will take place from state s to state s'.

Markov decision processes (MDPs) extend DTMCs by incorporating *nondeterministic choice* in addition to probabilistic behaviour. An MDP $(S, Act, Steps)$ comprises a set of actions Act and a (partial) probabilistic transition function $Steps : S \times Act \to Dist(S)$, which maps state-action pairs to probability distributions over the state space S. In each state $s \in S$, one or more distinct actions can be taken and, assuming that action $a \in Act$ is chosen, the distribution $Steps(s, a)$ gives the probability of making a transition to each state.

Stochastic two-player games (STPGs) generalise MDPs by allowing the nondeterministic choices in the model to be resolved by two distinct players. An STPG is a tuple $(S, (S_1, S_2), Act, Steps)$ where the set of states S is partitioned into two disjoint subsets S_1 and S_2. As for MDPs, $Steps : S \times Act \to Dist(S)$ is a function mapping state-action pairs to distributions over states. This is a *turn-based* game: in each state s of the game, either player 1 or player 2 selects an action $a \in Act$, depending on whether s is in set S_1 or S_2.

2.2 Probabilistic Model Checking & PRISM

Probabilistic model checking involves the construction and analysis of a probabilistic model. Usually, a high-level description language is used to model a system (here, we use the PRISM [16] modelling language). Then, one or more quantitative properties are formally specified and analysed on the model. Typically, *probabilistic temporal logics* are used to formalise properties. In this paper, we use PRISM's temporal logic-based query language, which is essentially the logic PCTL [12], extended to include reward-based properties [15,9]. This can be used to express properties of both DTMCs and MDPs. We also generalise the logic to capture properties of stochastic games.

PCTL extends the well known temporal logic CTL with a probabilistic (P) operator. Informally, this places bounds on the probability of the occurrence of certain events in the model. We will illustrate the use of PRISM temporal logic queries, using some simple examples, referring the reader to [12,15] for precise details of the syntax and semantics. For a DTMC, typical queries would be:

- $\mathtt{P}_{<0.01}[\,\Diamond\, fail\,]$ - "the probability of a failure occurring is less than 0.01"
- $\mathtt{P}_{\geq 0.95}[\,\Diamond\, end\,]$ - "the probability of the protocol terminating is at least 0.95".

For simplicity, we restrict our attention to *reachability* queries (in the examples above, $fail$ is a *label*, denoting a particular subset of the DTMC's states S and $\Diamond\, fail$ refers to

the event in which a state from this set is reached). In practice, we often use a *quantitative* variant of the $\mathtt{P}$ operator, denoted $\mathtt{P}_{=?}$, which returns the actual probability of an event's occurrence, e.g.:

– $\mathtt{P}_{=?}[\,\Diamond\, \mathit{fail}\,]$ - "what is the probability of a failure occurring?"

Whereas in a DTMC it is relatively straightforward to define the probability of an event such as $\Diamond\, \mathit{fail}$, for MDPs we must also take account of the nondeterminism in the model. The standard approach is to use the notion of *strategies* (also referred to as *policies*, *schedulers*, etc.). A strategy resolves nondeterminism in an MDP (i.e. chooses an action in a state), based on its execution history. For a specific strategy, we *can* define the probability of an event. Thus, probabilistic model checking focuses on best- or worst-case analysis, quantifying over all possible strategies. We still employ quantitative properties, which now ask for *minimum* or *maximum* probabilities:

– $\mathtt{P}_{\max=?}[\,\Diamond\, \mathit{fail}\,]$ - "what is the maximum probability of a failure occurring?"

For a stochastic game, the same approach generalises naturally, but we require strategies for both players. Usually, we assume that the two players have opposing objectives, for example player 1 aims to minimise the probability of $\Diamond\, \mathit{fail}$ and player 2 tries to maximise it. Extending the notation from above, we write:

– $\mathtt{P}_{\min,\max=?}[\,\Diamond\, \mathit{fail}\,]$ - "what is the minimum probability of failure that player 1 can guarantee, assuming that player 2 tries to maximise it?"

For this simple class of (zero-sum) properties, these values are well defined [7].

Finally, we also use properties based on *rewards*, which capture a variety of additional quantitative measures. For consistency across all three types of models, we assume a simple state-based scheme, i.e., a *reward function* $\rho : S \to \mathbb{R}_{\geq 0}$. We consider the *expected total reward* accumulated until some target set of states is reached. Consider a DTMC with reward function time and a label end denoting a set of target states. We write, for example:

– $\mathtt{R}^{\mathit{time}}_{=?}[\,\Diamond\, \mathit{end}\,]$ - "what is the expected time for the algorithm to complete?"

In exactly the same style as above, these queries generalise to MDPs and STPGs:

– $\mathtt{R}^{\mathit{time}}_{\max=?}[\,\Diamond\, \mathit{end}\,]$ - "what is the maximum expected algorithm completion time?"
– $\mathtt{R}^{\mathit{time}}_{\min,\max=?}[\,\Diamond\, \mathit{end}\,]$ - "what is the minimum expected time for algorithm completion that player 1 can guarantee, assuming player 2 tries to maximise it?"

PRISM [16] is a probabilistic model checker. It supports several different types of models, including DTMCs and MDPs (it also supports continuous-time Markov chains and probabilistic timed automata). On these models, a wide range of temporal logic-based properties can be checked, including all of those illustrated above. For the work presented here, we have built a prototype extension of PRISM that adds support for STPGs in the form of solving turn-based stochastic two-player zero-sum games (i.e. model-checking temporal formulae of the form described above). Models to be analysed by PRISM are described in a high-level modelling language based on guarded command notation; we discuss this further in Section 4.1.

3 Definitions and Algorithms

The purpose of this section is to provide definitions of terminology used throughout this paper and then present the algorithms which will be analysed.

3.1 Definitions

We introduce definitions of *agent organisations*, *tasks*, *teams*, and formulae for computing *rewards* to measure the performance of both individual agents and agent teams.

Definition 1 (Agent Organisation). *An* agent organisation *is a tuple* $O = \langle A, N, R, R_A \rangle$ *where:*

- $A = \{a_1, a_2, ..., a_n\}$ *is a set of agents,*
- $N = \{\{a_i, a_j\} : \text{“}a_i \text{ and } a_j \text{ are neighbours”}\}$ *is a neighbourhood relation,*
- $R = \{r_1, r_2, \ldots, r_k\}$ *is a set of resource types, and*
- $R_A = \{R_{a_1}, R_{a_2}, \ldots, R_{a_n}\}$ *is a set of agent resources where* $r_j \in R_{a_i} \iff$ *“agent* a_i *has a resource* r_j*”.*

Definition 2 (Task). *A* task $T_i = \{r_i : \text{“}r_i \text{ is required by the task } i\text{”}\}$ *is a set of resources that are required to accomplish* T_i*. By* $T = \{T_1, T_2, \ldots, T_t\}$ *we denote a collection of tasks.*

Definition 3 (Team). *A* team *of agents is denoted by* $M_i = \{a_j : \text{“}a_j \text{ is a member of team } i\text{”}\}$*, and the set of all teams is* $M = \{M_1, M_2, \ldots, M_m\}$*. By* $\bar{M} = \bigcup_{1 \leq i \leq m} M_i$*, we denote the set of all agents that are committed to some team. For* $1 \leq i \leq m$*,* $R_{M_i} = \bigcup_{a \in M_i} R_a$ *is the set of resources the team* M_i *has. The team* M_i *is said to be able to accomplish the task* T_j *iff* $T_j \subseteq R_{M_i}$*.*

Definition 4 (Rewards). *For agent* a*, we define two types of* reward*:*

- *Type* W_1*, which rewards the* agent *with 1 point if it is in the team which was able to complete its task after team formation is over; and 0 otherwise. Formally,*

$$W_1(a) = \begin{cases} 1 & \text{if } \exists M_i . a \in M_i \wedge T_i \subseteq R_{M_i}, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

- *Type* W_2*, which rewards 1 point to the* team *which was able to complete its task, and 0 otherwise. The reward is shared equally between team members.*

$$W_2(a) = \begin{cases} \frac{1}{|M_i|} & \text{if } \exists M_i . a \in M_i \wedge T_i \subseteq R_{M_i}, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

For a set of agents A*, the rewards are defined accordingly as the total reward achieved by its members, i.e.,*

$$W_1(A) = \sum_{a \in A} W_1(a) \qquad W_2(A) = \sum_{a \in A} W_2(a). \tag{3}$$

The underlying idea of these two types of rewards is that W_2 provides incentives for agents to form smaller teams which can accomplish tasks, whereas W_1 motivates agents to be in a successful team. From the organisation's perspective, the W_1 reward should be used when resources are limited, whereas the W_2 reward will encourage agents to introduce resource redundancy into teams, but this may ensure that tasks are completed with higher probabilities.

3.2 Algorithms

In this section we provide pseudocode for the algorithms which we later analyse. During the team formation process, each agent performs as follows: when it is not committed to any team (meaning that it is available and not assigned to any task), it considers each task in a random order. If a task currently has no other agents committed to it, the agent can choose to initialise a team, and does so with the probability given in Eqn. (4) (i.e., the ratio between neighbours that are not committed to any team and total number of neighbours).

$$IP_a = \frac{|\{a' \in A \,:\, \{a, a'\} \in N \wedge a' \notin \bar{M}\}|}{|\{a' \in A \,:\, \{a, a'\} \in N\}|}. \tag{4}$$

For team joining, if an agent is eligible for a team, it always joins the team. Note that only uncommitted agents can commit to a new or partially filled task, and committed agents can not decommit from a given task.

In Alg. 1 we reproduce pseudocode for the JOINTEAM algorithm introduced in [10]. This combines team initialisation and team joining. This algorithm will be modelled and analysed as a DTMC, as we shall see in Sec. 5.1.

Algorithm 1 JOINTEAM algorithm [10] (probabilistic and deterministic)

```
procedure JoinTeam(a, T, M)
    for all T_i ∈ T in random order do
        if a ∉ M̄ then                                              ▷ agent is not committed
            if |M_i| = 0 then                                      ▷ team for task i is empty
                if R_a ∩ T_i ≠ ∅ then   ▷ agent has skill (replaced by true if called from Online, cf. Alg. 3)
                    with probability IP_a: M_i ← M_i ∪ {a}         ▷ initialise a team (see Eqn. (4))
                end if
            else if ∃{a, a'} ∈ N.a' ∈ M_i then                     ▷ there is neighbour in team for task i
                if R_a ∩ T_i \ R_{M_i} ≠ ∅ then  ▷ agent has a missing resource (replaced by true if called from Online)
                    M_i ← M_i ∪ {a}                                ▷ join team
                end if
            end if
        end if
    end for
end procedure
```

To tackle the problem of finding the best team initialisation and team joining strategy, we modify the original JOINTEAM algorithm by allowing agents to make decisions regarding what actions to take, instead of picking one randomly. Technically, the changes are as follows, which are highlighted in Alg. 2.

– Allow agents to consider tasks in arbitrary order instead of randomly;

– Replace probabilistic choice to initialise the team by nondeterministic choice;
– Allow agent *not* to join a team even if it has a resource and neighbour in that team.

This algorithm allows analysis of the best-case performance that can be achieved by the protocol. It also allows us to analyse agent organisations with *hostile* agents, which aim to reduce organisation's performance. It will be modelled and analysed as an MDP and STPG respectively, as we shall see in Sec. 5.2 and Sec. 5.3. Furthermore, we ob-

Algorithm 2 JOINTEAM algorithm (non-deterministic extension)

```
procedure JoinTeam(a, T, M)
    for all T_i ∈ T in arbitrary order do
        if a ∉ M̄ then                                              ▷ agent is not committed
            if |M_i| = 0 then                                      ▷ team for task i is empty
                if R_a ∩ T_i ≠ ∅ then          ▷ agent has skill (replaced by true if called from Online, cf. Alg. 3)
                    M_i ← M_i ∪ {a} or M_i ← M_i                   ▷ initialise a team or do nothing
                end if
            else if ∃{a, a'} ∈ N.a' ∈ M_i then                     ▷ there is neighbour in team for task i
                if R_a ∩ T_i \ R_{M_i} ≠ ∅ then  ▷ agent has a missing resource (replaced by true if called from Online)
                    M_i ← M_i ∪ {a} or M_i ← M_i                   ▷ join a team or do nothing
                end if
            end if
        end if
    end for
end procedure
```

serve that there are two natural ways to call JOINTEAM: the OFFLINE procedure first initialises the set of tasks and then sequentially calls the JOINTEAM procedures of every agent in random order, as described in Alg. 1. In contrast, the ONLINE routine calls JOINTEAM procedures for agents *before* selecting the tasks. (The JOINTEAM algorithm needs to be adapted slightly, see Alg. 1, the 5th line.) We investigate both the offline and online versions of the algorithms because they provide a nice comparison between optimisation against specific tasks (offline), and distribution of tasks (online). As we will see in Sec. 5.3, whether the offline or online version results in better performance depends on network topology.

4 Models and Experimental Setup

4.1 PRISM Models

The PRISM model checker has been briefly described above. The purpose of this section is to explain how we model the algorithms from the previous section in PRISM. Due to space limitations we do not provide the source code of the models and properties used in this paper; instead, we have made them all available online[2]. Here we use a toy example from Fig. 1 to illustrate the design concepts. The system modelled in this example consists of two agents and a scheduling module which randomly generates the number of tasks to be performed (1 or 2, each with probability 0.5). Then, agents act in turn by choosing which team to join for each task. The reward structure *"total"*

[2] See `http://www.prismmodelchecker.org/files/clima11/`.

Algorithm 3 Offline and online versions of JOINTEAM algorithm

procedure OFFLINE(t) ▷ t - number of tasks
  $M = \{M_i = \emptyset \ : \ 1 \leq i \leq t\}$ ▷ initialise empty teams
  $T = \{T_i \neq \emptyset \ : \ T_i \subseteq_{random} R, \ 1 \leq i \leq t\}$ ▷ initialise tasks at random
  for all $a \in A$ in random order **do**
    JOINTEAM(a, T, M)
  end for
  perform tasks and compute rewards
end procedure

procedure ONLINE(t) ▷ t - number of tasks
  $M = \{M_i = \emptyset \ : \ 1 \leq i \leq t\}$ ▷ initialise empty teams
  for all $a \in A$ in random order **do**
    JOINTEAM(a, T, M)
  end for
  $T = \{T_i \neq \emptyset \ : \ T_i \subseteq_{random} R, \ 1 \leq i \leq t\}$ ▷ initialise tasks at random
  perform tasks and compute rewards
end procedure

rewards 0.3 points for each task for which agents joined different teams and 1.0 points when agents cooperate. The choice of teams for an agent is nondeterministic (i.e. the underlying model will be either an MDP or STPG), but it could be made probabilistic in a way similar to the scheduler's generation of tasks (and thus become a DTMC).

The same principle has been applied to the models that we used for experiments. Each agent is modelled as a module with Alg. 1 and Alg. 2 encoded as guarded commands. There is a scheduler module which has Alg. 3 implemented as guarded commands. The reward structures are also described according to definitions in Eqn. (1)-(3).

From this high-level description of the algorithm, our extension of PRISM then constructs the corresponding models: DTMC, MDP, or STPG. For STPGs, we also need to specify the split of model states into those controlled by players 1 and 2. This is done with two expressions over PRISM model variables, describing these sets.

4.2 Experimental Setup

For our experiments we mainly consider organisations consisting of five agents which are organised into four networks: fully connected, ring, star, and a network having one isolated agent. Each agent is assigned one resource, and there are three different resources available. For each network, we find the optimal resource allocation with respect to task generation described[3] below using DTMC model checking (see Sec. 5). These organisations are then fixed and used in all experiments.

Agent Organisations. We run experiments with a set of five agents $A = \{a_1, a_2, a_3, a_4, a_5\}$ and a set of three resources $R = \{r_1, r_2, r_3\}$ arranged into four different agent organisations O_{fc}, O_r, O_s, O_{ia} (see Fig. 2 for graphical representations of these).

[3] We have chosen this way of allocating resources in order to easily show how the performance can be improved by changing the strategy while keeping actions unchanged (i.e. compare DTMC and MDP models).

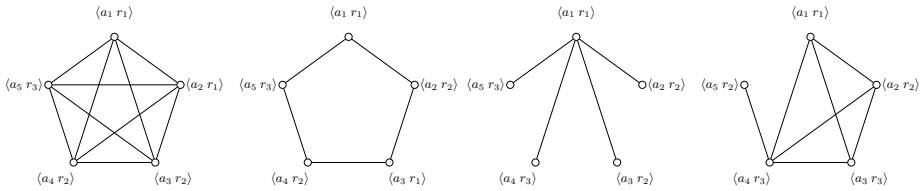
```
module scheduler
    turn : [1..3] init 1;
    num_tasks : [-1..2] init -1;
    [gen] num_tasks=-1 → 0.5 : (num_tasks′=1) + 0.5 : (num_tasks′=2);
    [go1] num_tasks>0 ∧ turn=1 → (turn′=2);
    [go2] num_tasks>0 ∧ turn=2 → (turn′=3);
    [do] num_tasks>0 ∧ turn=3 → (turn′=1) ∧ (num_tasks′=num_tasks − 1);
endmodule

module agent1
    team1 : [1..2] init 1;
    [go1] true → (team1′=1);
    [go1] true → (team1′=2);
endmodule

module agent2 = agent1 [go1=go2, team1=team2] endmodule

rewards "total"
    turn=3 ∧ team1≠team2 : 0.3;
    turn=3 ∧ team1=team2 : 1.0;
endrewards
```

Fig. 1: Example of a two agent system, described in PRISM's guarded command modelling language; see [16] for further details and links



(a) Fully connected ($\mathbf{O}_{fc}$) (b) Ring ($\mathbf{O}_r$) (c) Star ($\mathbf{O}_s$) (d) Isolated agent ($\mathbf{O}_{ia}$)

Fig. 2: Experimental configurations of the agent organisations with optimal resource allocation (see Tab. 2). In parentheses is the abbreviation that we will use to refer to the organisation throughout this paper.

Tasks. We fix seven different tasks that will be used in experiments $T = \{\{r_1\}, \{r_2\}, \{r_3\}, \{r_1, r_2\}, \{r_1, r_3\}, \{r_2, r_3\}, \{r_1, r_2, r_3\}\}$. When running the algorithm, two tasks T_1 and T_2 are picked uniformly and independently at random (with replacement) and are advertised to the agent organisation. So, there are a total of 49 different combinations of T_1 and T_2 that can be generated.

5 Experimental Results

In this section, we present results obtained using three models: DTMC, MDP, and STPG. Tab. 1 compares model construction information for different sizes of fully connected agent organisations[4]. All experiments are performed on a 2.8GHz Intel Core 2 PC, 4Gb of RAM running the Fedora Core 13 operating system. Nondeterministic

[4] We choose a fully connected agent organisation because it produces the largest models.

Agents	States	Transitions	Constr. Time (s)
2	1865	2256	0.1
3	17041	20904	0.3
4	184753	226736	3.4
5	2366305	2893536	74.4
6	35058241	42638400	2916.2

(a) DTMC

Agents	States	Transitions	Constr. Time (s)
2	1405	1846	0.1
3	9721	12474	0.2
4	76865	96664	1.1
5	731233	907992	5.1
6	8155873	10040112	29.7

(b) MDP and STPG

Table 1: Model comparison for different numbers of agents in a fully connected agent organisation for the offline version of Alg. 1.

Organisation O	Additional constraints	Example $\langle R_{a_1} R_{a_2} R_{a_3} R_{a_4} R_{a_5} \rangle$
O_{fc}	-	$R_A = \langle \{r_1\}\{r_1\}\{r_2\}\{r_2\}\{r_3\} \rangle$
O_r	$R_{a_1} \neq R_{a_5} \wedge \forall i < 5. R_{a_i} \neq R_{a_{i+1}}$	$R_A = \langle \{r_1\}\{r_2\}\{r_1\}\{r_2\}\{r_3\} \rangle$
O_s	$R_{a_1} = \{r\} \wedge \forall i > 1. r \notin R_{a_i}$	$R_A = \langle \{r_1\}\{r_2\}\{r_2\}\{r_3\}\{r_3\} \rangle$
O_{ia}	$R_{a_5} = \{r\} \wedge \exists i < 4. r \in R_{a_i}$	$R_A = \langle \{r_1\}\{r_2\}\{r_3\}\{r_3\}\{r_2\} \rangle$

Table 2: Optimal resource allocations with respect to rewards defined in Eqn. (3). All satisfy the constraint $\forall i. |R_{a_i}| = 1 \wedge \forall i. 1 \leq |\{R_{a_j} : r_i \in R_{a_j} (1 \leq j \leq 5)\}| \leq 2$.

models (MDPs/STPGs) have a smaller state space because agent choices do not have to be resolved at the model construction stage. However, the model checking is generally more time consuming for MDPs and STPGs than for DTMCs. The time needed for model checking is in the range of 5-240 seconds.

As mentioned in Sec. 4.2, for each topology from Fig. 2 we obtain optimal resource allocations using probabilistic model checking on the DTMC model of the offline version of the algorithm (see Alg. 3 and Alg. 1). The following PRISM temporal logic queries are used to compute the expected rewards of the agent organisation under a particular resource allocation:

– $\mathtt{R}^{W_j}_{=?}[\,\Diamond\, \mathit{finished}\,]$ - "what is the expected total reward W_j of an agent organisation when execution terminates?" ($j \in \{1, 2\}$).

After obtaining the expected rewards for all possible resource allocations, we selected the one with the highest expected reward. The results are summarised in Tab. 2. The resource allocations given in column "Example" of Tab. 2 will be used for all future experiments and are shown in Fig. 2. We decided to fix resource allocations in this way in order to show how model-checking techniques can be used to improve algorithm performance by synthesising strategies (see discussion of MDP results in Sec. 5.2).

5.1 DTMC Analysis

In this section, we present the results for model checking the DTMC model of Alg. 1 for experimental agent organisations from Fig. 2, as well as offline and online versions of the algorithm (see Alg. 3 for details).

O	$W_1(O)$	$\min_{a \in A} W_1(a)$	$\max_{a \in A} W_1(a)$
O_{fc}	2.54906	0.44958	0.75073
O_r	2.30359	0.35494	0.63985
O_s	1.87278	0.28677	0.72568
O_{ia}	2.38529	0.28867	0.68769

(a) Offline. W_1 reward structure.

O	$W_2(O)$	$\min_{a \in A} W_2(a)$	$\max_{a \in A} W_2(a)$
O_{fc}	1.49125	0.26721	0.42238
O_r	1.42923	0.23531	0.38625
O_s	1.16649	0.18582	0.42321
O_{ia}	1.43599	0.20621	0.39907

(b) Offline. W_2 reward structure.

O	$W_1(O)$	$\min_{a \in A} W_1(a)$	$\max_{a \in A} W_1(a)$
O_{fc}	3.53645	0.64101	0.97239
O_r	3.48638	0.55089	0.91190
O_s	2.52500	0.41934	0.84761
O_{ia}	3.37359	0.41186	0.93601

(c) Online. W_1 reward structure.

O	$W_2(O)$	$\min_{a \in A} W_2(a)$	$\max_{a \in A} W_2(a)$
O_{fc}	1.29743	0.24247	0.32657
O_r	1.31882	0.23157	0.31297
O_s	0.94404	0.16060	0.30158
O_{ia}	1.25560	0.17970	0.31990

(d) Online. W_2 reward structure.

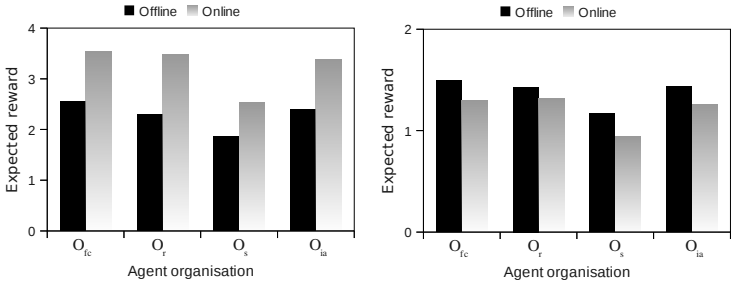
Table 3: Model checking results for agent organisations from Fig. 2 with optimal resource allocations from Tab. 2 for offline and online versions of Alg. 3. Tables also show largest and smallest individual agent rewards. For a histogram view of the total reward data, see Fig. 3.

Tab. 3 shows the results obtained for the expected rewards of agent organisations in different settings, namely, using W_1 and W_2 reward structures (see Eqn. (3) for organisations and Eqn. (1) and Eqn. (2) for individual agents), and offline and online versions of Alg. 1. The following PRISM temporal logic queries were used to obtain the results:

- $\mathtt{R}^{W_j}_{=?}[\,\Diamond\, \mathit{finished}\,]$ - "what is the expected total reward W_j?" ($j \in \{1, 2\}$),
- $\mathtt{R}^{W_j}_{=?}[\,\Diamond\, (\mathit{finished} \wedge a_i \in \bar{M})\,]$ - "what is the expected reward W_j for agent a_i?" ($j \in \{1, 2\}$, $i \in \{1, 2, 3, 4, 5\}$).

As can be seen in Tab. 3, agents organised in O_s have the worst expected rewards in all settings, and also the largest disparity between the worst and best performing individual agents. Both of these characteristics are not surprising because agent a_1, which is placed in the middle, is most likely to be in a winning team, whereas the others do not have any choice but to join the team with a_1. Organisation O_{ia}, which has one isolated agent, shows a smaller differences between the worst and the best performing agents, but this is only because the performance of the "best" agents is lower, whereas the "worst" agent's performance is very close to that of O_s.

Fig. 3 compares total rewards of all organisations in offline and online settings. It can be seen that a fully connected organisation O_{fc} has the best overall performance in all but the online version using the W_2 reward structure, where it is outperformed by O_r. It is interesting to note that a more significant difference between O_{fc} and O_{ia} only emerges when moving to the online setting. This shows that having one agent which is isolated does not affect the ability of the organisation to efficiently respond to the generated tasks, but impacts its chances to organise before the tasks are generated. This is where the advantages of O_r emerge: in the online setting, it not only starts outperforming O_{ia} with respect to overall performance and disparity, but gets very close



(a) Reward $W_1(O)$. (b) Reward $W_2(O)$.

Fig. 3: Expected rewards for agent organisations when using online and offline (see Alg. 3) versions of Alg. 1.

O	T_1 compl.	T_2 compl.	T_1 and T_2 compl.
O_{fc}	0.74562	0.74562	0.49596
O_r	0.71461	0.71461	0.47062
O_s	0.58324	0.58324	0.23639
O_{ia}	0.71799	0.71799	0.44839

(a) Offline

O	T_1 compl.	T_2 compl.	T_1 and T_2 compl.
O_{fc}	0.64871	0.64871	0.31320
O_r	0.65941	0.65941	0.36712
O_s	0.47202	0.47202	0.07465
O_{ia}	0.62780	0.62780	0.29270

(b) Online

Table 4: Task completion probabilities for optimal agent organisations using Alg. 1's offline and online versions (see Alg. 3).

to the performance of O_{fc} using the W_1 reward structure, and produces a better total W_2 reward while keeping similar disparity levels. An attentive reader would have noticed that the online version produces larger expected W_1, but smaller expected W_2 rewards. This observation shows that, in an online version, the algorithm organises agents into teams that increase the expectation for more agents to be in a successful team, but decrease the expected total number of tasks completed. This is summarised in Tab. 4, which shows the task completion probabilities. The following PRISM temporal logic queries were used to find the probabilities:

- $\mathtt{P}_{=?}[\,\Diamond\, T_j\_done\,]$ - "what is the probability to complete task T_j?" ($j \in \{1, 2\}$),
- $\mathtt{P}_{=?}[\,\Diamond\,(T_1\_done \wedge T_2\_done)\,]$ - "what is the probability to complete both tasks?".

Using formal analysis for DTMCs, we produced *exact* values of expectations for properties of the system (task completion probabilities and rewards were used as examples), so that even small differences between different organisations can be captured precisely. Also, we focused on one particular strategy defined in Alg. 1, but the PRISM code can be adapted easily to analyse other strategies and reward structures. In the next section we will explore how the performance can be improved by changing the strategy of the agents so that they collaborate to optimise the performance of the organisation.

O	T_1 compl.	T_2 compl.	T_1 and T_2 compl.
O_{fc}	1.0	1.0	0.67346
O_r	1.0	1.0	0.67346
O_s	0.82857	0.82857	0.39183
O_{ia}	1.0	1.0	0.67346

(a) Offline

O	T_1 compl.	T_2 compl.	T_1 and T_2 compl.
O_{fc}	1.0	1.0	0.42857
O_r	1.0	1.0	0.42857
O_s	0.88571	0.88571	0.12653
O_{ia}	1.0	1.0	0.42857

(b) Online

Table 5: Maximum task completion probabilities for optimal agent organisations using Alg. 2's online and offline versions (see Alg. 3).

5.2 MDP Analysis

In this section, we present the analysis of Alg. 2, which is modelled as an MDP. Using PRISM we find the maximum expected rewards and task completion probabilities for all agent organisations and compare the results with the strategy used in Alg. 1. This is achieved by synthesizing the optimal strategy for the MDP using PRISM and then model-checking the formulae on the resulting DTMC. Due to space limitations we do not present the actual strategies here.[5] In Tab. 5 we can see the maximum expected task completion probabilities that can be achieved. All organisations except O_s can ensure that at least one task is completed with probability 1.0, no matter what the scheduling is. The following PRISM temporal logic queries were used to get the results:

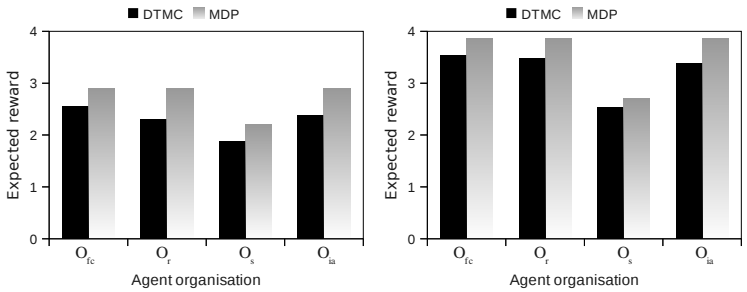
- $\mathtt{P}_{\max=?}[\,\Diamond\, T_j\_done\,]$ - "what is the maximum probability to complete task T_j?" ($j \in \{1, 2\}$),
- $\mathtt{P}_{\max=?}[\,\Diamond\,(T_1\_done \wedge T_2\_done)\,]$ - "what is the maximum probability to complete both tasks?".

Fig. 4 compares maximum expected rewards for Alg. 1 that can be achieved by all agents collaborating. It is not very difficult to see that O_{fc}, O_r and O_{ia} have the same maximum reward, no matter whether W_1/W_2 reward or online/offline version is taken. These outperform the star organization O_s in all circumstances. More significant improvement can be obtained for the offline version for both rewards than for the online version. This result shows that there is more potential for collaboration for agents in the offline version. The small performance improvement for the online version suggests that the original strategy of Alg. 1 is close to optimal when teams are formed before tasks are advertised.

The PRISM temporal logic queries used to find the rewards were the following:

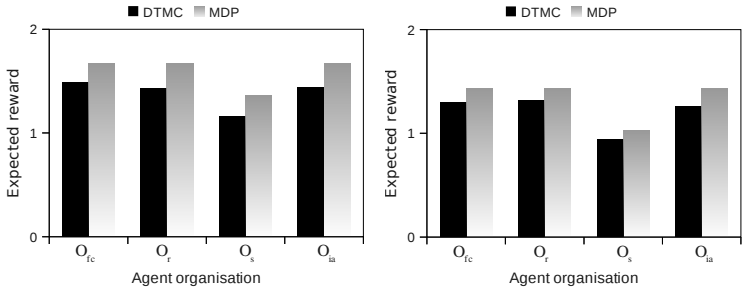
- $\mathtt{R}^{W_j}_{\max=?}[\,\Diamond\, \mathit{finished}\,]$ - "what is the maximum expected total reward W_j?" ($j \in \{1, 2\}$),
- $\mathtt{R}^{W_j}_{\max=?}[\,\Diamond\,(\mathit{finished} \wedge a_i \in \bar{M})\,]$ - "what is the maximum expected reward W_j for agent a_i?" ($j \in \{1, 2\}$, $i \in \{1, 2, 3, 4, 5\}$).

[5] The instructions on how to generate optimal strategies for MDPs can be found at:
http://www.prismmodelchecker.org/manual/RunningPRISM/Adversaries



(a) Offline. Reward $W_1(O)$.

(b) Online. Reward $W_1(O)$.



(c) Offline. Reward $W_2(O)$.

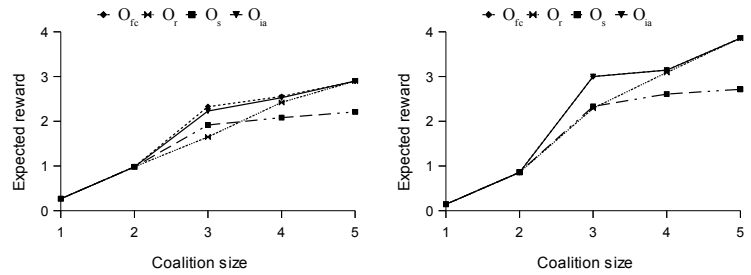
(d) Online. Reward $W_2(O)$.

Fig. 4: Bar charts comparing offline and online versions of Alg. 1 analysed as a DTMC with the best-case performance of Alg. 2 analysed as an MDP.

Using MDP model checking we have obtained the optimal strategy for the protocol and compared its performance to Alg. 1. The designer could choose to synthesize and use this strategy but it is worth noting that, even if the designer chooses not to implement this particular strategy, it still serves as a "best-case" benchmark for measuring other algorithms' performance. This analysis allows us to evaluate the effects of a fully collaborative behaviour of agents. However often only limited collaboration can be achieved. In order to facilitate analysis of such systems, one has to go beyond MDPs. In the next section we show how STPGs can be used for this purpose.

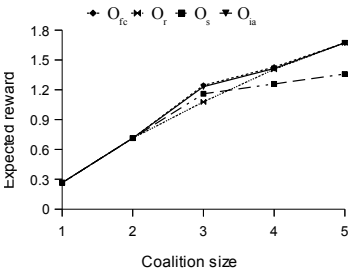
5.3 STPG Analysis

In this section we focus on the analysis of Alg. 2, but, in contrast to the previous section, we distinguish agents as either *cooperative* or *hostile*. This is naturally modelled as an STPG, by considering each class of agents (cooperative or hostile) as a separate player in a zero-sum game. Cooperative agents collaborate fully to act in the interest of the organisation, i.e., maximising rewards W_1 or W_2, whereas all hostile agents together take actions so as to minimise the expected rewards gained by the organisation. As a result of solving the STPG, we are able to obtain (synthesise) the optimal strategies for both cooperative and hostile agents, these can be used in further design of the system.

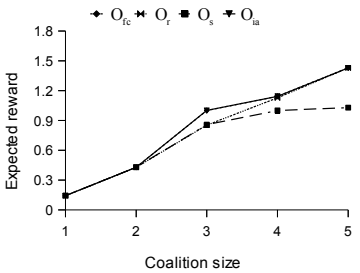


(a) Offline. Reward $W_1(O)$.

(b) Online. Reward $W_1(O)$.



(c) Offline. Reward $W_2(O)$.



(d) Online. Reward $W_2(O)$.

Fig. 5: Graphs comparing how the optimal coalition's performance depends on its size.

Here we present the expected rewards achieved by the optimal strategies, but due to space limitations we omit the strategies themselves.

We are mainly interested in finding the optimal coalition. In general, a coalition $C \subseteq A$ is a subset of agents who cooperate to ensure certain goals, irrespective of how the other agents behave in the organization. We consider two criteria: the largest probability to accomplish tasks, and the largest reward (W_1 or W_2) achieved by the coalition. To this aim, we use the following PRISM temporal logic queries which, as mentioned earlier, have been extended for STPGs.

- $\mathtt{P}_{\max,\min=?}[\,\Diamond\, T_j\_done\,]$ - "what is the maximum probability for coalition C to complete task T_j when agents in $A \setminus C$ are hostile?" ($j \in \{1,2\}$),
- $\mathtt{P}_{\max,\min=?}[\,\Diamond\, (T_1\_done \wedge T_2\_done)\,]$ - "what is the maximum probability for coalition C to complete both tasks T_1 and T_2 when agents in $A \setminus C$ are hostile?",
- $\mathtt{R}^{W_j}_{\max,\min=?}[\,\Diamond\, \mathit{finished}\,]$ - "what is the maximum expected reward W_j for coalition C when agents in $A \setminus C$ are hostile?" ($j \in \{1,2\}$).

For all agent organisations from Fig. 2, we enumerate all possible coalitions with different sizes and use PRISM to compute task completion probabilities (one task or both tasks) and rewards obtained (W_1 or W_2). These are done for both online and offline versions of the algorithm. It turns out that there exist coalitions of all sizes that are optimal with respect to all evaluated criteria; they are shown in Tab. 6. This result highlights the

O	1	2	3	4	5
O_{fc}	$\langle a_1 \rangle$	$\langle a_1, a_3 \rangle$	$\langle a_1, a_3, a_5 \rangle$	$\langle a_1, a_2, a_3, a_5 \rangle$	$\langle a_1, a_2, a_3, a_4, a_5 \rangle$
O_r	$\langle a_1 \rangle$	$\langle a_2, a_3 \rangle$	$\langle a_1, a_4, a_5 \rangle$	$\langle a_1, a_2, a_4, a_5 \rangle$	$\langle a_1, a_2, a_3, a_4, a_5 \rangle$
O_s	$\langle a_1 \rangle$	$\langle a_1, a_2 \rangle$	$\langle a_1, a_2, a_4 \rangle$	$\langle a_1, a_2, a_3, a_4 \rangle$	$\langle a_1, a_2, a_3, a_4, a_5 \rangle$
O_{ia}	$\langle a_1 \rangle$	$\langle a_1, a_4 \rangle$	$\langle a_1, a_2, a_4 \rangle$	$\langle a_1, a_2, a_4, a_5 \rangle$	$\langle a_1, a_2, a_3, a_4, a_5 \rangle$

Table 6: Optimal coalitions of all sizes for agent organisations from Fig. 2.

importance of positions in the network and resources held by the agents. For example, agent a_4 is in all optimal coalitions of sizes greater than 1 for O_{ia}. This is because it is connected to all agents, including agent a_5 which is isolated from other agents. For O_r, however, the structure of the optimal coalition varies depending on coalition size. For example, for size 2 the optimal coalition consists of agents a_2 and a_3, but neither of them is in the optimal coalition of size 3.

Fig. 5 shows a comparison of agent organisations in terms of the maximum performance for different coalition sizes. O_{fc} outperforms others in all examples. This is interesting, because it suggests that having strong connectivity within the team outweighs the exposure to many hostile agents. Performance of O_r is the most consistent, as the maximum reward increases steadily with the coalition size. However, to be as effective as more connected networks like O_{fc}, the coalition has to contain most agents in the network. Better performance of O_s against O_r for coalition sizes up to 3 illustrates the importance of having a highly interconnected agent for small coalitions.

Important differences between the online and offline settings can be seen for reward W_1 in Fig. 5a and 5b. When going from coalition size 2 to 3, especially, for strongly connected O_{fc} and O_{ia}, coalitions can ensure that one task is completed with probability 1.0 , and thus guaranteeing reward of at least 3 for the coalition, which results in a jump of performance.

In this section, we have shown how an extension of PRISM to support STPGs can be used to verify properties of the multi-agent system that can be enforced by particular agent coalitions. This competitive scenario allowed us to analyse the team formation algorithm from the coalitional perspective, finding the optimal coalitions and comparing their performance on different network topologies.

6 Conclusion and Future Work

In this paper, we have presented a comprehensive, formal analysis of a team formation algorithm using probabilistic model checking. We believe this demonstrates the strong potential of these techniques to the formal analysis of multi-agent systems and hope that this case study will serve as a motivation for further work in this area.

As ongoing and future work, we plan to explore parallel execution of the JOINTEAM algorithm for multiple agents, as here we only considered sequential execution, and consider the problem of synthesising optimal agent organisations for task distributions from a mechanism design perspective. We also plan to develop our prototype extension of PRISM into a fully-fledged model checker for stochastic games and to equip the analysis with abstraction techniques to allow for analysis of larger systems.

References

1. S. Abdallah and V.R. Lesser. Organization-based cooperative coalition formation. In *IAT*, pp. 162–168. IEEE, 2004.
2. T. Ågotnes, W. van der Hoek, and M. Wooldridge. Reasoning about coalitional games. *Artificial Intelligence*, 173(1):45–79, 2009.
3. R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
4. P. Ballarini, M. Fisher, and M. Wooldridge. Uncertain agent verification through probabilistic model-checking. In *Proc. of SASEMAS'06*, 2006.
5. E. Bonzon, M.-C. Lagasquie-Schiex, and J. Lang. Efficient coalitions in Boolean games. In *Texts in Logic and Games*, vol.5, pp. 293–297. 2008.
6. K. Chatterjee, T. Henzinger, and N. Piterman. Strategy logic. In *Proc. CONCUR 2007*, LNCS 4703, pp. 59–73. Springer, 2007.
7. A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.
8. P.E. Dunne, W. van der Hoek, S. Kraus, and M. Wooldridge. Cooperative boolean games. In *Proc. of AAMAS'08*, pp. 1015–1022. ACM, 2008.
9. V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated Verification Techniques for Probabilistic Systems. In *Proc. SFM'11*, Springer, 2011. To appear.
10. M.E. Gaston and M. desJardins. Agent-organized networks for dynamic team formation. In *Proc. of AAMAS'05*, pp. 230–237. ACM, 2005.
11. R. Glinton, P. Scerri, and K. Sycara. Agent-based sensor coalition formation. In *Proc. Fusion'08*, pp. 1–7. IEEE, 2008.
12. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
13. S. Kraus, O. Shehory, and G. Taase. Coalition formation with uncertain heterogeneous information. In *Proc. of AAMAS'03*, pp. 1–8. ACM, 2003.
14. M. Kwiatkowska and G. Norman. Verifying randomized Byzantine agreement. In *Proc. of FORTE'02*, *LNCS* 2529, pp. 194–209. Springer, 2002.
15. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *Proc. SFM'07*, LNCS 4486, pp. 220–270. Springer, 2007.
16. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. of CAV'11*, LNCS. Springer, 2011. To appear.
17. A. Lomuscio and F. Raimondi. MCMAS: A model checker for multi-agent systems. In *Proc. of TACAS'06*, LNCS 3920, pp. 450–454. Springer, 2006.
18. E. Manisterski, E. David, S. Kraus, and N.R. Jennings. Forming efficient agent groups for completing complex tasks. In *Proc. of AAMAS'06*, pp. 834–841. ACM, 2006.
19. M.J. Osborne and A. Rubinstein. *A course in game theory*. The MIT press, 1994.
20. O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, 1998.